**Česká zemědělská univerzita v Praze**

**Provozně ekonomická fakulta**

**Katedra informačních technologií**

# Diplomová práce

## Monitoring IT prostředků

**Adam Novotný**

# ČESKÁ ZEMĚDĚLSKÁ UNIVERZITA V PRAZE

Provozně ekonomická fakulta

# ZADÁNÍ DIPLOMOVÉ PRÁCE

Bc. Adam Novotný

Informatika

Název práce
**Monitoring IT prostředků**

Název anglicky
**Monitoring of IT resources**

---

**Cíle práce**

Práce bude tematicky zaměřena na problematiku automatizovaného monitoringu sloužící k přehlednému logování IT prostředků.
Hlavním cílem práce bude zanalyzovat možnosti monitoringu a následně navrhnout monitorovací řešení implementované pro vybraný podnik.
Dílčími cíli budou:

- charakterizovat problematiku automatizovaného monitoringu,
- vybrat prostředky, které se budou monitorovat,
- analyzovat možnosti monitorování,
- vybrat vhodné nástroje,
- provést návrh a implementaci monitorovacího řešení.

**Metodika**

Na základě studia odborných a vědeckých zdrojů bude zpracována teoretická část diplomové práce. Pomocí získaných poznatků bude objasněn pojem automatizovaného monitoringu a analyzovány možné nástroje pro návrh řešení.

V praktické části bude analyzován současný stav monitoringu ve vybraném podniku. Následně budou definované nedostatky a navrženy optimalizace. V dalším kroku budou vybrány vhodné nástroje, z kterých bude sestaveno řešení, které bude následně implementováno pro daný podnik. Závěrem bude dané řešení vyhodnoceno a formulovány závěry diplomové práce.

**Doporučený rozsah práce**

60-80 stran

**Klíčová slova**

monitoring, logy, Grafana, Prometheus, Docker container

---

**Doporučené zdroje informací**

BASTOS, Joel a Pedro ARAÚJO. Hands-On Infrastructure Monitoring with Prometheus: Implement and Scale Queries, Dashboards, and Alerting Across Machines and Containers. 1. Birmingham: Packt Publishing, Limited, 2019. ISBN 9781789612349.

HERNANDEZ, Rodrigo Juan. Building IoT Visualizations Using Grafana: Power up Your IoT Projects and Monitor with Prometheus, LibreNMS, and Elasticsearch. 1. Birmingham: Packt Publishing, Limited, 2022. ISBN 9781803231938.

MULI, Joseph. Beginning DevOps with Docker: Automate the Deployment of Your Environment with the Power of the Docker Toolchain. 1. Birmingham: Packt Publishing, Limited, 2018. ISBN 9781789532401.

SABHARWAL, Navin a Piyush PANDEY. Monitoring Microservices and Containerized Applications: Deployment, Configuration, and Best Practices for Prometheus and Alert Manager. 1. New York: Apress L. P., 2020. ISBN 9781484262153.

SMITH, Roderick W. Linux Essentials. 1. Toronto: John Wiley & Sons, Incorporated, 2012. ISBN 9781118106792.

---

**Předběžný termín obhajoby**

2023/24 LS – PEF

**Vedoucí práce**

Ing. Michal Stočes, Ph.D.

**Garantující pracoviště**

Katedra informačních technologií

Elektronicky schváleno dne 4. 7. 2023

**doc. Ing. Jiří Vaněk, Ph.D.**

Vedoucí katedry

Elektronicky schváleno dne 3. 11. 2023

**doc. Ing. Tomáš Šubrt, Ph.D.**

Děkan

V Praze dne 28. 11. 2023

**Čestné prohlášení**

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

V Praze dne 31.3.2024 _____

**Poděkování**

Rád bych touto cestou poděkoval Ing. Michalu Stočesovi, Ph.D. za pomoc a vedení diplomové práce. Rád bych také poděkoval mojí rodině za podporu během celého mého studia. V neposlední řadě patří poděkováním také firmě SABO Mobile IT s.r.o.

# Monitoring IT prostředků

**Abstrakt**

Tato diplomová práce pojednává o automatizovaném monitoringu IT prostředků v souvislosti s využitím v podnicích. Monitoring je v dnešním ICT prostředí stále se rozvíjející metodou, která podstatně usnadňuje firmám jejich procesy a tím i zlepšuje jejich ekonomickou situaci.

V této práci je úvodem představena problematika monitoringu v teoretické části. V rámci toho jsou nejprve uvedeny obecné informace jako výhody monitoringu, jeho odlišná možná provedení, představení standardů, vysvětlen pojem automatizace. Dále jsou také popsány technické části, se kterými se následně pracuje ve vlastní práci. Jedná se zejména o Linux server s jeho různými službami nebo možnostmi provedení a o containerizaci, především pomocí platformy Docker. Závěrem teoretické části je zpracována analýza dostupných monitorovacích nástrojů, která je základem pro výběr do architektur ve vlastní práci.

Ve vlastní práci je nejprve představen vybraný podnik. Pro ten jsou následně vytvořeny analýzy pojednávající o současném stavu. Na základě těchto analýz jsou stanoveny požadované use cases, díky kterým jsou navrženy architektury monitorovacích řešení. Tato stanovená řešení jsou následně implementována do produkčního prostředí podniku.

U celých implementovaných řešení je v poslední části ověřena jejich funkčnost, vysvětlen význam získávaných hodnot a provedeno zhodnocení.

**Klíčová slova:** monitoring, automatizace, Grafana, Prometheus, Linux, Docker container, metriky, logy, alerty

# Monitoring of IT resources

**Abstract**

This thesis deals with the automated monitoring of IT resources in the context of their use in companies. The monitoring is a growing method in today's ICT environment, which significantly simplify processes of companies, improving also their economic situation.

In this thesis the topic of monitoring is firstly introduced in the theoretical part. At first that includes general information such as the advantages of monitoring, its different possible implementations, the introduction of standards, and the concept of automation. The technical parts which are used in the practical part are described next. Especially those are Linux server with its different kinds of services or implementations and containerization mainly using the Docker platform. At the end of theoretical part the analysis of the available monitoring tools are also done. That is the basis for the selection into architectures in the practical part.

In the practical part the selected company is introducted at first. The analysis for this company about its current status are created next. Based on those analyses are determinated the required use cases, which leads to designed architectures of the monitoring solutions. Those defined solutions which are implemented into the production environment of company.

In the last part, the functionality of implemented solutions is verified, the meaning of the values are explained and the evaluation is performed.

**Keywords**: monitoring, automation, Grafana, Prometheus, Linux, Docker container, metrics, logs, alerts

# Obsah

# 1 Úvod

Monitoring IT prostředků je velice důležitou součástí moderního prostředí ICT technologií. Dokáže nahrazovat spoustu manuální práce, předcházet nežádoucím incidentům a poskytovat možnost analýz systémů. V dnešní době však automatizovaný monitoring stále není brán napříč spektrem, nejen IT podniků, za samozřejmost. To je ovšem škoda, neboť monitoring poskytuje nesporné výhody, které by se měly ve většině případů firmám velice vyplatit. Efektivní monitoring by měl být schopný v průběhu času relativně rychle zaplatit náklady na jeho vybudování a následně poskytovat možnost snížení nákladů nebo zvýšení zisků, což by mělo vést ke zlepšení ekonomické situaci podnikům, které monitoring efektivně využijí.

Diplomová práce se věnuje právě této problematice monitoringu IT prostředků a snaze poskytnout bližší vhled do uvedené problematiky. Tento bližší vhled by měl být jakýmsi spouštěčem pro podniky, které tak doposud neučinily, pro rozhodnutí vybudovat vlastní monitoring IT prostředků. V této práci je vybrán také jeden konkrétní podnik, kde je monitoring IT prostředků implementován, což může být vhodný příklad i pro jiné podniky v případě obecných řešení, jak tento monitoring vybudovat.

Samotná diplomová práce se skládá ze tří částí. Teoretická část, vlastní praktická práce a závěrečné zhodnocení vypracovaného řešení.

V teoretické části práce je nejprve charakterizována problematika automatizovaného monitoringu. Nejdříve jsou uvedeny obecné informace o monitoringu IT prostředků a následně uvedeno jeho možné dělení. Příklady dělení a různých typů monitoringu jsou uvedeny zejména z toho důvodu, že existuje obrovské množství variant, jak lze monitorovat. Základem je pak vybrat tu správnou v souvislosti s potřebami daného podniku. Další významnou částí monitoringu v teoretické části je uvedení jeho možné automatizace. To je v souvislosti s monitoringem IT prostředků nutně vyžadovaný aspekt, který by měl být co nejvíce zdokonalen pro eliminaci manuální práce. Dále jsou představeny konkrétní IT prostředky, se kterými se bude pracovat následně ve vlastní práci. Je tedy nutné mít obecný přehled o těchto prostředcích. Poslední částí teorie je zpracování přehledu a analýzy dostupných monitorovacích nástrojů, což vede k možnosti výběru konkrétních nástrojů pro následnou vlastní práci.

V této vlastní práci je nejprve představen vybraný podnik SABO Mobile IT s.r.o. pro navržení a implementaci automatizovaného monitoringu IT prostředků. Následně je analyzován stav používaných prostředků a nástrojů pro lepší možnost výběru monitorovacích řešení. Dále je analyzován stav samotného využití monitoringu, který již firma v malé míře a poměrně neefektivně využívala. Na základě těchto analýz byly stanoveny use cases, které udávají konkrétní potřeby firmy na vybudování monitoringu. To v kombinaci s poznatky z teoretické části, především analýzy monitorovacích nástrojů, vede k následnému stanovení monitorovacích řešení za využití několika odlišných architektur. Tato řešení jsou následně jednotlivě implementována přímo do produkčního prostředí firmy.

V závěrečné části je vytvořené a implementované řešení zhodnoceno především faktorem času a zkušenostmi, neboť dané řešení je již nějakou dobu produkčně nasazené. Nejprve je ověřena funkčnost a zhodnocena spolehlivost implementovaných architektur. Dále je vysvětlen význam získávaných hodnot pomocí tohoto monitoringu a je uvedeno jejich možné využití ve firmě. Na základě těchto poznatků je následně zpracováno ekonomické zhodnocení, avšak bez možnosti uvedení konkrétních částek. Toto zhodnocení je tedy uvedeno pouze v rámci předpokládaného nárůstu zisků nebo poklesu nákladů ke každému z variant monitoringu.

# 2 Cíl práce a metodika

## 2.1 Cíl práce

Práce bude tematicky zaměřena na problematiku automatizovaného monitoringu sloužící k přehlednému logování IT prostředků.

Hlavním cílem práce bude zanalyzovat možnosti monitoringu a následně navrhnout monitorovací řešení implementované pro vybraný podnik.

Dílčími cíli budou:

- charakterizovat problematiku automatizovaného monitoringu,
- vybrat prostředky, které se budou monitorovat,
- analyzovat možnosti monitorování,
- vybrat vhodné nástroje,
- provést návrh a implementaci monitorovacího řešení.

## 2.2 Metodika

Na základě studia odborných a vědeckých zdrojů bude zpracována teoretická část diplomové práce. Pomocí získaných poznatků bude objasněn pojem automatizovaného monitoringu a analyzovány možné nástroje pro návrh řešení.

V praktické části bude analyzován současný stav monitoringu ve vybraném podniku. Následně budou definované nedostatky a navrženy optimalizace. V dalším kroku budou vybrány vhodné nástroje, z kterých bude sestaveno řešení, které bude následně implementováno pro daný podnik. Závěrem bude dané řešení vyhodnoceno a formulovány závěry diplomové práce.

# 3 Monitoring IT prostředků

## 3.1 Definice

Monitoring IT prostředků je systematický proces průběžného sledování, zaznamenávání (logování) a analyzování technických parametrů a celkové chování daných prostředků s cílem zajištění jejich bezproblémového a nepřetržitého provozu, výkonu, bezpečnosti a dalších klíčových funkcí. Jedná se o klíčovou součást správy IT, která umožňuje organizacím okamžitě reagovat na problémy, plánovat zdroje a optimalizovat IT služby podle aktuálních potřeb. Základem monitoringu je tedy shromažďování dat z různých zdrojů, jako jsou servery, databáze, síťová zařízení, aplikace a další pomocí specializovaných nástrojů a SW. Tato data bývají následně analyzována pro identifikaci trendů, výkonnostních úzkých míst, možných výpadků a bezpečnostních incidentů. V praxi se o správu monitoringu běžně stará oddělení DevOps.

Z technologického hlediska monitoring zahrnuje nástroje a procesy, kterými lze měřit a sledovat IT systémy. Avšak monitoring je mnohem více než to. Poskytuje překlad mezi metrikami generovanými danými prostředky a obchodní hodnotou. Monitorovací systém tyto metriky převádí do UX/UI přívětivého prostředí pro možnost lepšího porozumění. Toto měřitelné UI poskytuje zpětnou vazbu obchodu tak, aby zajistil efektivní dodávání toho, co zákazníci chtějí. UI také poskytuje zpětnou vazbu IT oddělení, které z toho může vyčíst, co nefunguje, neposkytuje dostatečnou kvalitu služeb nebo zkrátka neodpovídá stanoveným požadavkům. Monitoring tak bývá konstruovaný zejména ze dvou důvodů, ty můžeme nazvat jako obchod (určený zákazníkům) a informační technologie (určený spravující IT organizaci). [1]

### 3.1.1 Monitoring pro obchod

V tomto případě je zákazníkem monitorovacího systému klient. Monitoring je vytvářen za účelem podpory již dodané aplikace nebo jiného ICT řešení, nebo samostatně konfigurovaný na již existující infrastrukturu nebo aplikaci klienta, se kterou organizace nemá nic společného. Klade se zde důraz hlavně na dobrý UX/UI, který umožňuje klientovi činit správná rozhodnutí ohledně jeho infrastruktury, produktů, aplikací a jiných technologií. [1]

Monitorování může také zprostředkovávat klientovi měřitelnou obchodní hodnotu, kterou technologie přináší. To může být dobře měřitelné a analyzované například ze záznamů počtu návštěv, zaznamenání počtu určitých kroků v aplikaci, vytíženosti dané technologie a dalších.

Dalším možným případem využití monitoringu pro obchod může být zasílání pravidelných reportů o technickém stavu aplikace, kterou podnik dodává klientovi. Zde se může objevit například průběh využití IT prostředků za dané časové období a jiné důležité informace o chodu aplikace.

### 3.1.2 Monitoring pro informační technologie

Druhého zákazníka můžeme nazvat jako IT, tedy organizaci, která dané řešení vytváří, zejména DevOps tým a ostatní lidé, kteří spravují a udržují infrastrukturu včetně jejích technologií pro organizaci. Zde má monitoring za účel zprostředkovat informace o technickém stavu infrastruktury vlastním zaměstnancům, zejména pro možnost efektivního plánování vlastních zdrojů. Monitorování je možné také intenzivně využít k detekci, diagnostice a řešení závad a v technologickém prostředí organizace. To přispívá mnoha daty, která mohou být dobrým podkladem pro kritická rozhodnutí ohledně produktů a technologií a měří dosažení cílů těchto projektů. Je tedy klíčovou součástí cyklu řízení, vztahu s interními produkty a pomáhá ukázat, že peníze podniku jsou vynaloženy dobře. Podnik bez monitoringu nemůže v dnešním světě IT odvádět dobrou práci.[1]

### 3.1.3 Význam monitoringu

Význam monitoringu v IT prostředí se neodvíjí pouze od jeho definice, ale především od přínosů, které monitoring organizacím přináší. Jak již bylo zmíněno monitoring umožňuje IT týmům (především DevOps) udržovat systémy a aplikace v optimálním provozním stavu, což přispívá k stabilnímu a efektivnímu chodu celé organizace. Představuje tedy zásadní roli ve správě a optimalizaci informačních technologií. Jeho možná aplikace se rozkládá přes široké spektrum možných variant – od zajištění kontroly bezpečnosti a výkonu sítí a serverů, přes monitorování aplikací a služeb až po analýzu a optimalizaci procesů. Díky monitoringu mohou organizace dosáhnout vysoké úrovně provozní efektivity, minimalizovat rizika výpadků a zabezpečit tak kontinuitu provozu a celého obchodu.

Další výhodou a důvodem proč monitoring implementovat jsou jednoznačně finance a ekonomika. Efektivní monitorovací strategie umožňuje organizacím snížit provozní náklady tím, že předchází nákladným opravám v důsledku výpadků. Cena za vybudování monitorovací infrastruktury bývá opravdu minoritní v porovnání s cenou výpadků aplikací v produkčním prostředí. I kdyby se nemělo jednat přímo o výpadek, může být pro podnik monitoring finančně výhodný vzhledem k možnosti efektivního plánování a optimalizaci využití zdrojů. Například detekce a řešení kritických míst v systému může vést k lepšímu využití infrastruktury bez nutnosti dalších investic. Dále monitorování umožňuje podnikům lépe plánovat budoucí investice díky přesnějším datům o využití a potřebách.

Monitoring také přispívá k zajištění vyšší uživatelské spokojenosti. V dnešní době, kdy jsou digitální služby nezbytnou součástí každodenního života, je schopnost rychle identifikovat a řešit problémy s výkonem nebo dostupností klíčová pro udržení pozitivního vztahu se zákazníky. Monitoring poskytuje cennou možnost nahlédnutí do toho, jak skuteční uživatelé interagují s aplikacemi a službami, což umožňuje organizacím neustále vylepšovat své produkty a služby.

Další výhodou je zlepšení bezpečnostního prostředí. V současné době je počet kybernetických hrozeb a celkově útoků stále vyšší. Monitoring tak může poskytovat klíčovou obrannou strategii prostřednictvím neustálého sledování a analýzy definovaných bezpečnostních událostí, jako například requesty na server, případně z jakých adres. Tím umožňuje IT správcům DevOps rychle identifikovat a reagovat na potenciální bezpečnostní incidenty dříve, než mohou způsobit významné škody.

V neposlední řadě může být význam a použití monitoringu dosti specifické a neobvyklé, jako například možnost napomáhání k dodržování regulativních požadavků a norem. V mnoha zemích a odvětvích jsou organizace povinny sledovat a reportovat o svých IT systémech a datových tocích. Systémy pro monitorování poskytují nástroje potřebné k zajištění toho, že organizace splňuje tyto požadavky a zároveň je možné pomocí monitoringu tyto reporty vytvářet. To je zásadní pro minimalizaci právních a finančních rizik.

Z těchto důvodů je jasné, že monitoring IT prostředků není jen technologickou nutností, ale strategickou investicí, která umožňuje organizacím udržet si konkurenční výhodu, zajišťuje vysokou kvalitu služeb a produktů, minimalizuje rizika spojená s IT provozem a tím dokáže šetřit firmám zdroje i finance. Avšak v dnešní době stále ještě není

monitoring napříč podniky vnímán jako nezbytná součást vývoje. To je věc, která by se měla časem změnit a monitoring IT prostředků by se měl dostat do povědomí všem IT firmám.

## 3.2 Dělení monitoringu

Monitorovat lze v rámci IT spoustu různých věcí několika odlišnými metodami. Proto se zde podíváme na možné dělení monitoringu. Toto dělení není vždy přesně striktně dané, jedná se spíše o lepší porozumění problematice a představě kde a jak se dá monitoring využít.

### 3.2.1 Podle monitorovaného prvku

Zde je monitoring rozdělen dle toho, jakou metriku či prvek vybíráme. Jedná se o individuální rozdělení, proto by typů mohlo být ještě více, avšak zde jsou vypíchnuty hlavní typy, které lze v IT prostředí monitorovat.

#### 3.2.1.1 Výkon

Prvním a zároveň nejčastějším výstupem monitoringu bývá výkon zdroje, který je monitorován. Nejčastěji sledovanými metrikami jsou tyto tři: CPU, RAM, disk, avšak je možné setkat se i s dalšími v závislosti na sledovaném zařízení. Může zde být například i GPU, podkategorie již zmíněných jako třeba disk read, disk write, CPU load, utilization, steel time a další. Právě tyto metriky bývají velice zásadní a často sledované z důvodu možného předcházení výpadkům v důsledku nedostatečných prostředků, nebo naopak šetření zdrojů a tím možné šetření i financí při zjištění nadměrné kapacity systému oproti potřebám.

Konkrétně pak u CPU, ač se to na první pohled nezdá, je možné běžně sledovat hned několik aspektů. Od běžného procentuálního využití 0 % – 100 %, přes CPU load běžně sledovaný u UNIX based systémů. CPU load je matematická definice ukazující množství práce, které CPU provádí jako procento z celkové kapacity. Každý proces, který čeká na CPU, zvyšuje zátěž o 1 a proces, který je obsluhován naopak snižuje zátěž o 1. Load average (průměr zatížení) je míra, kolik úkolů čeká ve frontě na spuštění jádra po určitou dobu. U Linuxových systémů je možné se běžně setkat se třemi hodnotami. Průměr za jednu minutu, průměr za 5 minut a průměr za 15 minut. Hodnoty se zde pohybují od 0,0, což znamená že je systém úplně nečinný, až po celá čísla dle počtu procesorových jader systému.

Pokud by se hodnota dostala až na tato celá čísla, může se jednat o přetížený systém. Dále CPU Utilization neboli využití CPU. Zde se jedná o dobu, kdy je CPU v nečinném módu. Využívá se jako ukazatel toho, jak moc je CPU v dané chvíli vytížené. Poslední zmíněnou podmnožinou možnosti sledování CPU se zaměřením na Linuxový systém je CPU steal time. To je procento času, kdy virtuální procesor čeká na fyzický procesor, zatímco hypervisor obsluhuje jiný virtuální procesor. To se děje ve virtualizovaných prostředích, jako jsou například AWS, Azure, GCP. [2]

Další již zmíněnou a velice důležitou možností monitoringu je operační paměť RAM. Ta slouží k ukládání informací, které programy a celkově procesy potřebují, když jsou spuštěny. Memory představuje úložiště dat, které umožňuje přístup k uloženým datům v libovolném pořadí, a to náhodně, nikoli pouze postupně. Oproti tomu jiné typy paměťových zařízení, jako jsou klasické disky, mohou většinou přistupovat k datům na úložném médiu pouze v předem určeném pořadí. To je dáno omezeními v jejich mechanickém provedení. Není to ale vyloženě pravidlem, zejména u OS Linux existuje možnost vytváření takzvaného swapu. Zde se jedná o rozšíření fyzické RAM o část pevného disku nebo SSD. Když operační systém vyčerpá dostupnou RAM, vyměňuje data mezi RAM a swapovým prostorem. Tento mechanismus, který zvyšuje efektivitu správy paměti, je známý jako swapping. Můžeme zde tedy monitorovat jak klasickou memory, tak i swap, což může znovu předejít pádu systémů z důvodu jeho vyčerpání. Lze zde opět použít několik možných metrik jako memory usage, memory free, memory cache a tak dále. [3,4]

Posledním z již zmíněných prvků výkonu je tedy disk. I v tomto případě se opět jedná v rámci monitorování o možnost předcházení vyčerpání disku a tím zamezení přerušení chodu systému. HW disku může mít podobu například HDD elektromagnetických disků, které jsou zpravidla pomalejší. Ty však mohou šetřit náklady, pokud není vyžadováno rychlé čtení nebo zápis dat. V dnešní době jsou však nejpoužívanější SSD disky, které jsou sice dražší, avšak poskytují rychlejší zápis a čtení. Zde můžeme monitorovat buďto klasické metriky jako disk usage, disk available, disk used, tak i specifičtější metriky jako disk latency, disk read, disk write, disk IOPS a další. [5]

#### 3.2.1.2 Dostupnost

Monitorování dostupnosti neboli uptimu je jedním z nejprimitivnějších, avšak nejzásadnějších ukazatelů daného systému. Jedná se o měření doby, po kterou je server,

webová stránka či jiný IT systém v provozu (tzv. up). Zjednodušeně řečeno, pokud je daný prostředek často nedostupný (tzv. down), pak je hodnota downtime vysoká, a naopak uptime nízký (pokud by se hledělo na průměrnou hodnotu v čase). Samozřejmě právě u této metriky je oproti jiným sledovaným prvkům dost pravděpodobné si všimnout, zda je daný prostředek up nebo down i bez monitoringu, avšak to už může být pozdě. Zde si můžeme vytvořit například alertovací systém s notifikacemi do mobilu všech lidí, kteří mají s nasazením daného prostředku co do činění. To umožní sjednání okamžité nápravy a znovu nasazení systému tak, aby se co nejvíce minimalizovaly škody a ušlé zisky. Navíc je následně možno s takovouto metrikou vytvářet analýzy a reporty za dané časové období. [6]

Uptime může být vyjádřen několika způsoby. Ať už jako procentuální hodnota za určité časové období, tak například jako aktuální hodnota 1 (up) / 0 (down), nebo jako boolean hodnoty například true / false.

Jsou zde tři důležité faktory ovlivňující dobu provozu datového centra. Těmi jsou spolehlivost HW, dostupnost a servisovatelnost (neboli RAS). V případě selhání serveru může pomoci redundantní server jako záloha udržovat danou aplikaci stále up. Další známou strategií, jak udržet vysoký uptime, je klastrování serverů. To zajišťuje vysokou dostupnost IT služeb. Skupina propojených serverů, nazývaná serverový klastr, spolupracuje na zlepšení vyvážení zátěže systému (tzv. load balancing), dostupnosti služeb a výkonu. V případě selhání serveru mohou jiné servery v klastru převzít danou aplikaci a její load ze serveru který selhal. [6]

Monitoring se zde dá vytvářet několika způsoby. Nejběžnějším způsobem je odesílání automatizovaných http requestů na konkrétní URL v přednastavených časových intervalech, kde se následně kontrolují odpovědi neboli responzy. Existují také API přednastavené pro kontrolu uptime. Určený časový interval pak závisí na konkrétních potřebách klienta či organizace. Obvykle se může pohybovat od 30s po 10min dle vážnosti dané aplikace a možností vzhledem ke zdrojům. [6]

#### 3.2.1.3 Síťový provoz

Monitoring síťového provozu se zaměřuje na analýzu a sledování datového toku v síti. Konkrétně se jedná o proces sběru, analýzy a reportování dat o síťovém provozu za účelem diagnostiky problémů se sítí, optimalizace výkonu nebo jednoduše pro lepší pochopení celkové aktivity v síti. Monitoring networku je nezbytnou součástí správy sítě

jako takové. Pomáhá předcházet problémům a může být také použit k diagnostice stávajících problémů. Například dle webu Teramind, Amazon úspěšně zmírnil útok DDoS v únoru 2020 s využitím jejich služby AWS Shield. [7]

Možných metrik pro sběr networkingových dat v rámci monitoringu je opět několik. Nejběžnější může být network I/O (input, output), kde se jedná o součet celkového datového toku směrem zvenčí do serveru i ze serveru ven (běžně v B/s). Dalšími metrikami pak mohou být například latency měřící rychlost toku dat, než dorazí ze serveru k jejich cílové destinaci po síti (běžně v ms), jiter, packet loss, network availability, network error rate, network response time a další. [8]

Důvodů použití monitoringu síťového provozu je hned několik. Prvním může být diagnostika síťových problémů a errorů. Když nastane problém se sítí, monitoring může pomoci určit zdroj problému pro možnost sjednání rychlé nápravy. Dle webu Teramind CISCO nedávno integrovalo dva nové monitorovací nástroje pro podporu jejich IT týmů zvládat každodenní problémy, které ovlivňují jejich síťové prostředí. Dalším důvodem může být zajištění dodržování regulačních požadavků. Pokud podnik podléhá konkrétním regulacím, jako je HIPAA nebo PCI DSS, může být dokonce i monitorování síťového provozu vyžadováno, aby bylo zajištěno dodržování těchto předpisů a norem. V neposlední řadě může být důvodem také ochrana před malwarem a dalšími hrozbami. Monitoring zde může identifikovat neobvyklou aktivitu, která by mohla signalizovat ohrožení malwarem nebo jinou bezpečnostní hrozbu. Web Teramind uvádí, že dle FBI počet stížností na ransomware se mezi lety 2019 a 2021 zvýšil o 82 %. Jeden konkrétní útok na skupinu Oregon Anesthesiology přimělo vedení zajistit nepřetržité monitorování, aby se předešlo dalším útokům. [7]

#### 3.2.1.4 Bezpečnost

Část z monitoringu bezpečnosti byla již zmíněna v předchozí kapitole *3.2.1.3 Síťový provoz*, neboť jsou spolu tyto dva typy monitoringu dosti spjaté. Avšak o nich rozhodně nemůžeme mluvit jako o totožných. Monitoring bezpečnosti (též nazýváno jako kybernetická bezpečnost) je automatizovaný proces sběru dat, která má za úkol odhalovat potenciální bezpečnostní hrozby. Se správným nastavením by pak měl monitoring bezpečnosti prioritně zasílat informace ohledně těchto hrozeb lidem, kteří se v rámci organizace starají o bezpečnost (běžně DevOps) tak, aby organizace mohla okamžitě

reagovat a tím předejít možným škodám, které mohou být fatální. Monitoring bezpečnosti je možné rozdělit na dvě části, a to SIM a SEM nebo jejich kombinaci SIEM. První ze zmíněných SIM je zaměřen na dlouhodobé shromažďování bezpečnostních logů a dat. Cílem je poskytnout historický přehled a reporty bezpečnostních událostí, což usnadňuje audit, dodržování předpisů a může být také využit pro vytváření analýz. Druhý typ SEM se soustředí na aktuální real-time monitorování s možností již zmíněného alertingu. Umožňuje organizacím rychle identifikovat a reagovat na bezpečnostní incidenty a hrozby v reálném čase. Poslední typ SIEM kombinuje funkce SIM a SEM. Poskytuje komplexní řešení pro monitorování bezpečnosti tím, že shromažďuje a analyzuje bezpečnostní data z různých zdrojů v reálném čase a zároveň umožňuje dlouhodobé uchování pro analýzu dat, historický přehled, audit a dodržování předpisů. [9, 10]

Samotné řešení monitoringu bezpečnosti pak může být implementováno dvěma způsoby. Prvním je endpoint monitoring neboli monitoring koncových bodů. Ten sleduje a kontroluje koncové body v síti, což jsou fyzická zařízení jako servery, PC, smartphony nebo VM. Pomáhá organizacím udržovat přehled o síti a může tak zabránit přerušení kontinuity aplikace nebo jiných obchodních operací způsobených nedostatkem konektivity nebo bezpečnostními problémy. Druhou variantou je monitoring celého networku, což bylo z velké části představeno v předešlé kapitole. Monitoring networku zahrnuje sledování a analýzu síťové aktivity k detekci a reakci na problémy s výkonem, které by mohly naznačovat průnik nebo zranitelnost sítě pro útok. [11]

Konkrétních významů monitoringu bezpečnosti je spoustu. Snížení rizika úniku dat, což se může dotýkat jak klientských nebo uživatelských dat tak i dat organizace. Funguje také jako prevence zpoždění a výpadků, kdy rychlá reakce a plán kontinuity mohou zabránit zpožděním a přerušením provozu, které by mohly nastat v důsledku porušení dat. Dále zajištění souladu s předpisy, kde pravidelný monitoring sítě, přísná kontrola přístupu a rozvoj komplexní bezpečnostní politiky přispívají k dodržování regulací a pomáhají chránit před potenciálními porušeními a sankcemi. A v neposlední řadě také slouží jako prevence neoprávněného přístupu. V dnešní době, kdy mnoho organizací funguje na dálku a využívá cloudové služby, je nezbytné implementovat opatření pro kontrolu přístupu, aby se zabránilo neoprávněným pokusům o přístup k datům.

#### 3.2.1.5 Databáze

Dalším možným prvkem, který lze monitorovat je databáze. Monitorování databází je proces, který zahrnuje nepřetržité sledování a analýzu datových toků a operací v rámci databázových systémů. Jeho hlavním účelem je identifikace a prevence potenciálních bezpečnostních hrozeb, zajištění plynulého a efektivního přenosu dat, ochrana citlivých informací a případně více specifické úkoly dle požadavků. Monitorování umožňuje IT týmům získat hlubší porozumění aktivit probíhajících v databázi jako například transakce, změny v konfiguraci, přístup uživatelů nebo využití zdrojů. Tento proces využívá různé nástroje a techniky ke sběru dat, a ta poskytují cenné informace o stavu a výkonu databází. [12]

Monitorování databází je klíčové pro zajištění vysoké dostupnosti, spolehlivosti a bezpečnosti databázových systémů. Pomáhá předcházet datovým poruchám a ztrátám tím, že umožňuje IT týmům identifikovat a řešit problémy dříve, než mohou způsobit významné škody nebo výpadky. Díky tomu mohou organizace rychle reagovat na abnormality, optimalizovat databázové procesy a zajistit, že jejich databázové systémy jsou v souladu s platnými předpisy a standardy. Efektivní monitorování také přispívá k optimalizaci výkonu databáze. To může být zajištěno tím, že odhaluje úzká místa a umožňuje jejich rychlou nápravu. Umožňuje také lépe plánovat budoucí potřeby zdrojů a infrastruktury na základě analýzy trendů využití dat. Tímto způsobem se monitorování databází stává také skvělým nástrojem pro podporu rozhodovacích procesů a strategického plánování v oblasti IT. [12]

Proces monitorování databází zahrnuje několik klíčových kroků. Na začátku stojí sběr a analýza dat, které zahrnují sledování dostupnosti zdrojů, včetně CPU, memory a disku a také síťového provozu, jak již známe z předchozích kapitol. Dalším krokem je monitoring databázových operací DAM. Pro ten se dají použít specializované nástroje a platformy, které poskytují real-time přehled o všech aspektech databázového prostředí. Tyto systémy umožňují například v rámci bezpečnosti automatické detekování neobvyklé nebo podezřelé aktivity, jako jsou pokusy o SQL injection nebo neautorizovaný přístup k datům. Dále může sloužit pro debuggování databází a systémů napojených na databázi, kde je možné také vytáhnout logy (záznamy) přímo z databáze umožňující nahlédnutí, co se v databázi odehrává. Mimo tyto SW je však možné dle specifických požadavků použít také svoje

vlastní řešení, jako např. scripty. Kromě již zmíněných scénářů může databázový monitoring poskytovat také pravidelné auditování a generování reportů, které pomáhají splňovat požadavky na dodržování předpisů a usnadňují správu databázové bezpečnosti.

#### 3.2.1.6 Aplikace

Posledním zmíněným typem v této práci je monitorování aplikací (APM). Většina dílčích částí monitoringu aplikací zde již byla zmíněna a APM představuje z většiny komplexní kombinaci předešlých zmíněných typů. Avšak i zde se najdou části monitoringu, které jsou od ostatních odlišné. APM tedy představuje specializovanou oblast monitoringu IT prostředků, zaměřenou přímo na sledování a optimalizaci výkonu aplikací. Na rozdíl od již zmíněného tradičního monitoringu, který se většinou zaměřuje na celou infrastrukturu a její dílčí části, APM poskytuje detailní pohled na výkon a dostupnost aplikací z pohledu koncového uživatele. Tím umožňuje IT týmům nejen identifikovat a řešit problémy s výkonem, ale také zlepšovat UX/UI daných aplikací.

Konkrétními body, které zde můžeme sledovat, mohou být například doba odezvy aplikace, výkon kódu aplikace, sledování a logování errorů. Tyto body se zaměřují především na BE, případně FE. Mimo nich to může být také již zmíněné uživatelské rozhraní UX/UI. Nebo je možné sledovat také závislosti aplikací mezi sebou a jejich funkčnost a kvalitu propojení. [13]

### 3.2.2 Podle zdroje, který je monitorován

Další variantou, dle které je možné dělit monitoring, je podle cílového zdroje který chceme monitorovat. Opět je možné toto rozdělení pojmout různými způsoby, avšak pro tuto práci jsou vybrány konkrétní tři možnosti toho co lze monitorovat. Těmito možnostmi jsou server, container, cloud. U všech se mohou objevovat stejné prvky monitoringu, avšak s určitými odlišnostmi nebo jinak implementované.

#### 3.2.2.1 Server

Monitoring serverů je nezbytným nástrojem pro zajištění stabilního a efektivního provozu IT infrastruktury. Jeho hlavním účelem je sledování a analýza výkonu, dostupnosti a využití zdrojů serverů. To jeho administrátorům umožňuje identifikovat a řešit potenciální problémy dříve, než dojde k jejich eskalaci. Jak již bylo zmíněno, tento proces zahrnuje

shromažďování klíčových metrik, jako je využití CPU, RAM, disku, networkingu a další. Umožňuje předvídání budoucích problémů na základě historických dat. Monitoring serverů je tedy zásadní pro zajištění vysoké dostupnosti a spolehlivosti IT služeb, neboť na těchto serverech, ať už se jedná o vlastní infrastrukturu fyzických serverů nebo VPS, běží veškeré služby a aplikace organizace. Umožňuje IT týmům (většinou DevOps) proaktivně identifikovat problémy a zajistit, že serverová infrastruktura je optimálně a bezpečně využívána. Kromě zajištění operativní efektivity a bezpečnosti, monitoring serverů také pomáhá snižovat náklady tím, že automatizuje procesy a úkoly, čímž umožňuje týmům soustředit se na úkoly, které vyžadují jejich odborné znalosti. Správa serverů tedy zahrnuje různé funkce, které se liší v závislosti na IT setupu organizace a počtu spravovaných serverů. Zahrnuje denní monitorování, aktualizace softwaru, instalaci nové technologie, řešení problémů a řešení vznikajících problémů. Součástí správy serverů je také plánování kapacity a zajištění, aby organizace měla dostatečné systémové zdroje pro podporu svých operací. Správa serverů v obou prostředích, fyzických i virtuálních, musí zahrnovat jak SW, tak i HW . [14, 15]

Existují však určité výzvy spojené se správou serverů, zejména v prostředí VPS, které emulují funkce fyzických serverů v sdíleném SW prostředí. VPS se staly populárními, protože umožňují organizacím lépe využít kapacitu svého HW a snížit náklady. Tyto virtuální servery zároveň vyžadují méně správy, údržby a bezpečnostních kontrol než fyzické servery, což také významně přispívá k úsporám nákladů. Organizace si běžně kupují na bázi měsíčního předplatného VPS od specializovaných poskytovatelů, kteří provozují infrastruktury obsahující tisíce fyzických serverů umístěných v datových centrech po celém světě. Těmito poskytovateli mohou být například Contabo, Forpsi, Wedos a spousta dalších. [15]

#### 3.2.2.2 Container

Dalším zdrojem, který můžeme monitorovat, je container. Zde je zmíněn vztah monitoringu k containeru, avšak k containeru jako takovému je více zmíněno v kapitole níže. Efektivní monitorování containerů jako jsou Docker, Kubernetes atp. je nezbytné pro správu moderních aplikací, neboť v dnešní době je containerizace čím dál využívanější metodou nasazení daných aplikací. Krátce lze zmínit, že containery jsou základní stavební bloky pro vývoj a nasazení aplikací v microservices architekturách a přinášejí množství

výhod jako zejména škálovatelnost, agilita a izolace zdrojů. Avšak s těmito výhodami přicházejí i výzvy spojené s jejich dynamickou povahou, které ztěžují tradiční přístupy k monitorování. [16, 17]

Prvním krokem efektivního monitorování containerů je pochopení, proč je jejich monitoring tak důležitý. Celý proces práce s containery, jako jejich vytváření, nasazování, restartování nebo naopak mazání, je velice rychlý. To umožňuje vysokou míru dynamiky a flexibility ve vývoji a provozu aplikací. Tato dynamika však přináší překážky, které je potřeba pro jejich možné řešení monitorovat. Těmito prvky může být klasické sledování využití zdrojů, zabezpečení, healthchecking, dostupnost a další. Sledování těchto prvků přispívá nejen k hladkému chodu a optimalizaci aplikací, ale umožňuje i redukci nákladů. Efektivní monitorování tedy umožňuje IT týmům rychle identifikovat a řešit problémy, optimalizovat výkon a zdroje, zajišťovat nepřetržitou dostupnost a spolehlivost aplikací a další funkce. To zahrnuje nejen sledování výkonu a zdrojů na úrovni samotných containerů, ale také pochopení vlivu containerů na celkový výkon a zdraví hostovaných systémů a serverů. [17, 18]

Monitoring containerů vyžaduje hloubkové porozumění nástrojům a technikám specifickým pro jejich prostředí. Ať už porozumění hostovanému prostředí pro možné získání klasických dat, jako jsou využití CPU, RAM, disk, logy (záznamy aplikace), nebo třeba uptime. Tak i porozumění dané aplikaci a jejímu kódu například pro trasování transakcí. Kromě toho, uspořádání containerů pomocí nástrojů jako již zmíněný Docker nebo třeba Kubernetes je také potřeba znát. V případě Kubernetes je přidána další vrstva komplexnosti, vyžadující monitorování klastrů a služeb pro správu škálovatelnosti a zajištění dostupnosti. Sběr a analýza aplikačních specifických metrik, jako jsou síťová latence, počet požadavků nebo úniky paměti, poskytují další možnou vrstvu dat, která jsou klíčová pro optimalizaci výkonu a zajištění kvality služeb. Přístupy k monitorování mohou zahrnovat využití vestavěných nástrojů, jako jsou příkazy Dockeru (docker logs, docker stats), stejně jako integraci s pokročilými monitorovacími řešeními, jako je Prometheus pro sběr metrik a Grafana pro vizualizaci dat. O tom je však více zmíněno v kapitolách níže. Pro účinné monitorování je zásadní zvolit řešení, která poskytují komplexní pohled na aplikaci a umožňují korelaci událostí a logů pro rychlou identifikaci a řešení problémů. Důležitá je také schopnost detailně analyzovat jednotlivé komponenty a vrstvy aplikace, snadná

integrace sledování do vývojového procesu a konfigurace alarmů a automatizace pro proaktivní řízení výkonu a dostupnosti. [16, 17, 18]

#### 3.2.2.3 Cloud

Monitoring cloudových služeb představuje proces sledování, hodnocení a správy těchto služeb kterých může být obrovské množství typů. Proto bývá monitoring velice specifický a často pro každého poskytovatele cloudových služeb odlišný. Více ke cloudům a cloudovým službám jako takovým je zmíněno opět v kapitole níže, tato kapitola je zaměřena na vztah monitoringu ke cloudovým službám. Tento proces může zahrnovat skoro všechny typy monitoringu, jako monitoring infrastruktury, sítí, výkonu aplikací, bezpečnosti, databází, nákladů nebo třeba dodržování předpisů. Každý z těchto typů se zaměřuje na specifické aspekty cloudového prostředí a přináší podrobný pohled na zdraví, výkon a jeho bezpečnost. [19]

Monitorovací nástroje určené pro cloudy jsou klíčové pro optimalizaci nákladů, zlepšení viditelnosti výkonu, benchmarking, zvýšení bezpečnosti a zajištění škálovatelnosti. Tyto nástroje pomáhají organizacím sledovat využití zdrojů, odhalovat bezpečnostní hrozby a možnost na ně reagovat, porovnávat výkon aplikací před a po upgradu infrastruktury. Pomocí cloudového monitoringu je možné také spravovat škálování zdrojů pro udržení efektivního výkonu a dostupnosti služeb. Většina cloudových poskytovatelů nabízí vlastní monitorovací služby, jako jsou Amazon CloudWatch pro AWS, Google Cloud Operations Suite pro Google Cloud Platform a Azure Monitor pro Microsoft Azure. Tyto služby poskytují možnost efektivního monitoringu většiny služeb, které tyto platformy nabízejí. Pro podniky s infrastrukturou využívající více cloudových poskytovatelů mohou být užitečné nástroje schopné sbírat logy a metriky ze všech cloudových prostředí najednou. V takovém případě je však často řešením vytváření vlastních monitorovacích řešení jako scripty s http requesty a podobně. Tato monitorovací řešení cloudových služeb by zároveň měly extrahovat pouze relevantní informace, standardizovat formáty a indexovat pro efektivní vyhledávání. [20]

Dobře zpracovaná řešení cloudového monitoringu zahrnují monitorování všech poskytovatelů, jak již bylo zmíněno, ale i služeb, které daná organizace používá. Zde se může jednat o SaaS, IaaS, PaaS a další druhy služeb, které poskytovatelé cloudu nabízejí. Opět je zde vhodné a nejjednodušší využití předdefinovaných nástrojů, které mohou měřit

výkon, bezpečnost nebo například chování zákazníků. To zahrnuje také pravidelné testování těchto monitorovacích nástrojů, aby se zajistilo, že jsou plně funkční v případě poruchy nebo bezpečnostního incidentu, a využití dat k odstranění problémů a implementaci oprav včas. Klíčem je výběr správného monitorovacího nástroje, který může snadno škálovat s rostoucí aktivitou organizace a nabízí jednoduchou instalaci a minimální údržbu. Ne vždy je však možné použít předdefinovaná řešení, a i zde se často musí přejít k řešením sestaveným na míru dle požadavků organizace. [21]

### 3.2.3 Další možnosti

Jak již bylo zmíněno na začátku kapitoly Dělení monitoringu, je zde opravdu veliké množství faktorů, podle kterých lze monitoring dělit. Nikde však není uvedeno přesné oficiální dělení monitoringu, což z něj dělá dosti subjektivní a diskutabilní téma. Můžeme zde však ještě zmínit další možnosti typů monitoringu, jako například dělení dle způsobu monitorování, metody sběru dat, podle rozsahu působnosti. Možností dělení by se určitě dalo najít ještě více avšak tyto způsoby pro bližší pochopení daného tématu postačí. Zároveň se zde dost možností také z části nebo úplně překrývá.

První zmíněná metoda dělení podle způsobu monitorování umožňuje dělit monitoring na aktivní a pasivní, přičemž každý z těchto přístupů má své specifické využití a výhody. Aktivní monitoring se zaměřuje na proaktivní sledování výkonu sítí, aplikací a infrastruktury pomocí synteticky generovaných dat. Tento přístup umožňuje simulaci uživatelského a síťového chování, aby se předem identifikovaly potenciální problémy a optimalizoval výkon před tím, než by uživatelé pocítili jakýkoli vliv na jejich práci. Aktivní monitoring je obzvláště vhodný pro testování QoS, identifikaci potenciálních problémů aplikací, hodnocení výkonu nového HW a sledování síťového výkonu. Díky své prediktivní povaze může aktivní monitoring identifikovat problémy, než skutečně ovlivní koncové uživatele. To umožňuje IT týmům provádět preventivní úpravy. Pasivní monitoring, na druhé straně využívá pouze reálná naměřená data k měřením a analýzám. Tento přístup poskytuje komplexní a podrobný pohled na skutečný výkon, protože používá skutečná data zaznamenaná ze síťového provozu a uživatelské aktivity. Pasivní monitoring je ideální pro monitorování stavu a zdraví serverů a sítě, identifikaci trendů v uživatelském chování, personalizaci uživatelské zkušenosti a upozornění na problémy, které vyžadují okamžitou pozornost. Díky použití reálných dat pasivní monitoring poskytuje hlubší přehled

o celkovém výkonu sítě a může identifikovat složitější a přerušované problémy. Oba přístupy, aktivní i pasivní monitoring, mají své místo v komplexní strategii monitoringu IT prostředí. Aktivní monitoring umožňuje týmům být o krok napřed při identifikaci a řešení potenciálních problémů, zatímco pasivní monitoring nabízí cenný náhled do reálného využití a výkonu. Pro dosažení nejlepších výsledků je doporučeno kombinovat oba přístupy, čímž se získá nejúplnější obraz o stavu a výkonu IT prostředí a aplikací. [22, 23]

Další možností je tedy dělení podle rozsahu metody sběru dat, což umožňuje monitoring rozlišit na monitoring založený na agentech a monitoring nepoužívající tyto agenty. Monitoring založený na agentech nabízí instalaci SW agenta přímo na monitorovaném zařízení. Nainstalovaný agent pak umožňuje shromažďování dat o výkonu, protokolech nebo třeba zobrazování konfiguračních informací, které pak odesílají na monitorovací server nebo platformu pro zobrazení a analýzu. Tento přístup poskytuje velmi podrobný pohled na sledovaný systém, umožňuje hlubší analýzu a nabízí větší kontrolu nad sběrem dat. Výhodou je také větší odolnost vůči problémům s konektivitou, neboť agenti mohou data ukládat lokálně a odeslat je, jakmile je obnoveno síťové spojení. Nevýhodou může ale být potřeba instalace, údržby a aktualizace těchto SW agentů, což může být zvláště ve velkých prostředích náročné. Zároveň ne všechny monitorovací nástroje a způsoby monitoringu tyto agenty nabízejí. Proto existuje i tzv. bezagentový monitoring. Ten naopak využívá standardní protokoly a vlastnosti monitorovaných systémů pro sběr dat na dálku, aniž by bylo nutné instalovat na cílovém systému dodatečný software. Mezi běžně používané protokoly patří HTTP, SNMP, WMI, ICMP (Ping) nebo JDBC. Tento přístup umožňuje sledování širokého spektra zařízení a systémů bez nutnosti jakékoli instalace speciálního SW, což zjednodušuje škálování a snižuje operační zátěž. Obecně jde o méně náročnou formu monitoringu vzhledem ke zdrojům sledovaných zařízení, ale nabízí méně podrobných dat než monitoring založený na agentech. Hlavní nevýhodou je omezení dat dostupných přes standardní protokoly, což může vést k menší flexibilitě ve sběru a analýze dat a často k potřebě vytváření vlastních scriptů pro získání specifických dat dle požadavků organizace. [24, 25]

Poslední zmíněnou možností je dělení podle rozsahu působnosti, což již z velké části bylo zmíněno u kapitoly dělení podle zdroje, který je monitorován. Zde se může jednat právě o dělení monitoringu na infrastrukturální monitoring, který z většiny zastupují servery, monitoring containerů a microservices nebo cloudových poskytovatelů a jejich služeb. Zde

se pokaždé jedná pouze o jeden ze sektorů monitorovaný zvlášť a je možné tedy říct, že toto celé je jedna skupina velice konkrétně zaměřená na daný sektor. Což umožňuje vysokou míru specifičnosti monitoringu ke každému prvku, avšak každý monitoring zvlášť nám nedává komplexní vhled do celého systému. Tento komplexní pohled je často pro identifikaci problému obzvláště potřeba. Proto je tu také end-to-end monitoring, který nabízí komplexní pohled na celý tok dat a požadavků uživatelů skrze různé systémy a služby. Tento typ monitoringu je nezbytný pro pochopení UX a pro identifikaci jakýchkoli problémů, které by mohly celý systém negativně ovlivnit. End-to-end monitoring umožňuje IT týmům rychle reagovat na problémy a zajišťuje, dostupnost služeb a reakcí v souladu s očekáváními uživatelů. [26]

## 3.3 Standardy

Mimo konkrétní technické záležitosti je důležité zmínit také standardy týkající se ať už přímo nebo nepřímo monitoringu IT prostředků. Ne ve všech organizacích je na standardy kladen velký důraz, avšak někde tomu tak může být a může se tak jednat o jasně daná pravidla, která jsou klientem nebo organizací vyžadována. Proto je vhodná jejich znalost. Jedná se o soubory osvědčených postupů, směrnic a pravidel, které určují, jak efektivně monitorovat a spravovat IT prostředí, včetně sítí, aplikací, HW a bezpečnostních systémů. Cílem těchto standardů je zajistit, aby byly systémy spolehlivé, bezpečné a efektivní, a to prostřednictvím systematického sběru a analýzy dat. Standardy IT monitoringu ucelují tato pravidla organizacím do jasné struktury, která by se měla dodržet. Mimo zmíněných technických záležitostí se může jednat například i o normy a pravidla vzhledem ke koncovým uživatelům.

Je dobré vypíchnout jednu z největších společností zaštiťující většinu těchto standardů a často také oficiálních definic týkajících se ICT prostředí. Tato společnost se nazývá zkráceně NIST, tedy. Národní Institut Standardů a Technologií. Institut zřídilo ministerstvo obchodu Spojených států roku 1901. Jedná se o vědecké laboratoře s účelem rozvoje inovací a průmyslové konkurenceschopnosti pomocí zlepšování vědeckého měření, standardů a technologií.

Jedním z klíčových standardů, které se přímo týkají oblasti IT monitoringu, je právě od již zmíněné společnosti NIST SP 800-137. Tento standard se konkrétně týká monitoringu bezpečnosti. Poskytuje směrnice pro nepřetržité monitorování informační bezpečnosti ve

federálních informačních systémech a organizacích. Tento dokument pomáhá organizacím vyvíjet strategii nepřetržitého monitoringu a implementovat program, který zajišťuje viditelnost aktiv organizace, povědomí o hrozbách, zranitelnostech a viditelnost efektivity nasazených bezpečnostních kontrol. Cílem je zajistit, že plánované a implementované bezpečnostní kontroly jsou v souladu s tolerancí organizace vůči riziku a poskytovat informace potřebné k včasné reakci na rizika, pokud pozorování naznačují, že bezpečnostní kontroly jsou nedostatečné. [27]

Dalším významným zdrojem v oblasti kybernetické bezpečnosti je NIST Cybersecurity Framework, který NIST vyvíjí s cílem poskytnout standardy, směrnice, osvědčené postupy a další zdroje pro potřeby amerického průmyslu, federálních agentur a širší veřejnosti. Tyto aktivity zahrnují produkci konkrétních informací, které organizace mohou okamžitě uvést do praxe, až po dlouhodobější výzkum, který předvídá pokroky v technologiích a budoucí výzvy. Některé úkoly NIST v oblasti kybernetické bezpečnosti jsou definovány federálními zákony, prezidentskými nařízeními a politikami, například úřadem pro řízení a rozpočet, který vyžaduje, aby všechny federální agentury implementovaly standardy a směrnice NIST pro systémy nevyžadující národní bezpečnost. [28]

Mimo to jsou zde již zmíněné standardy, které přímo nesouvisejí s monitoringem jako takovým, avšak v určitých směrech se jich i monitoring může dotýkat. Proto je dobré se seznámit i s několika nejznámějšími standardy celkově. Jedním z nich je například série ISO/IEC 27000, která poskytuje doporučení nejlepších postupů pro řízení informační bezpečnosti a správu informačních rizik prostřednictvím bezpečnostních kontrol v rámci celkového systému řízení informační bezpečnosti (ISMS). Tato série je navržena pro rozsáhlé využití, pokrývá nejen soukromí a důvěrnost. Je aplikovatelná na organizace všech velikostí a typů. Zahrnuje kontinuální zpětnou vazbu a vylepšování činností pro reakci na změny v hrozbách, zranitelnostech nebo dopadech těchto hrozeb, což jsou relevantní informace i pro monitoring. [29]

Standardů je tedy obrovské množství. Další, které by mohly stát za zmínku i vzhledem k monitoringu IT prostředků, mohou být například ITIL (Information Technology Infrastructure Library). Jak již z názvu vyplývá, jedná o knihovnu neboli sadu osvědčených postupů pro efektivní správu a poskytování IT služeb, zaměřenou na zlepšování kvality IT služeb a jejich souladu s obchodními potřebami. Další rámec, který je

dobré zmínit, je COBIT (Control Objectives for Information and Related Technologies). Tento rámec je vytvořen pro správu a řízení podnikové IT, který poskytuje model procesů a kontrolní mechanismy pro zajištění souladu a efektivity IT operací. Poslední rámec zmíněný v této práci, který by se mohl také dotýkat monitoringu, je GDPR (General Data Protection Regulation). Zde se jedná o nařízení Evropské unie o ochraně osobních údajů, které stanovuje pravidla pro zpracování a ochranu osobních údajů subjektů v EU. Pokud se tedy bude jednat o monitoring akcí koncových uživatelů, může být i GDPR zásadní nařízení, které je třeba dodržet. [30, 31]

## 3.4 Automatizace

Jedním z nejzásadnějších bodů monitoringu IT prostředků je automatizace. Ta se v dnešním světě IT stává základním kamenem, který umožňuje organizacím efektivně spravovat jejich rozsáhlé a složité technologické prostředí. Tento proces nahrazuje manuální, opakující se a časově náročné úkoly automatizovanými procesy, což vede k výraznému snížení možnosti lidských chyb a výrazně zvyšuje produktivitu. Automatizace zahrnuje širokou paletu činností, jako například automatizované nasazování (deploy) aplikací, správa konfigurací, bezpečnostní aktualizace, automatizované recovery po haváriích a mnoho dalších scénářů. Klíčem k úspěchu je schopnost monitorovacích systémů neustále automatizovaně monitorovat IT prostředí, identifikovat potřebné úpravy a případně provádět změny v reálném čase, což zajišťuje, že IT infrastruktura je vždy optimalizována pro aktuální potřeby organizace. [32]

Důležitost automatizace v IT nelze podceňovat. V dnešním rychle se vyvíjejícím světě ICT a celkově digitálním světě je schopnost rychle reagovat na jakékoli podněty zásadní pro udržení konkurenceschopnosti. Automatizace umožňuje IT týmům uvolnit čas a zdroje, které by jinak byly věnovány rutinním úkolům, a místo toho se mohou soustředit na strategické iniciativy, které podporují obchodní cíle nebo jiné činnosti, které nevyžadují repetitivní manuální práci. Navíc automatizace výrazně zvyšuje také spolehlivost IT operací tím, že minimalizuje riziko chyb způsobených lidským faktorem a zajišťuje, že všechny procesy jsou prováděny konzistentně a v souladu s nejlepšími postupy. [33]

Monitoring hraje v automatizaci zásadní roli několika způsoby. Za automatizaci je možné považovat už jen získávání dat a informací real-time. Tyto informace mohou být nezbytné pro konání rychlých rozhodnutí. Role automatizace zde však může být dotažena

ještě dál a tím víc omezit repetitivní lidskou činnost. Konkrétně tak, že i prvek rozhodnutí nebo reakce v závislosti na obdržené informace z monitoringu může být automatizován. Jednalo by se tedy o celý automatizovaný proces v rámci monitoringu od zjištění chyby po její vyřešení bez nutnosti zásahu, ba dokonce povšimnutí člověka. Ve spojení s pokročilými nástroji pro monitorování IT prostředků, automatizace umožňuje organizacím sledovat výkon, dostupnost a bezpečnostní stav jejich systémů v reálném čase. Díky tomu lze okamžitě identifikovat a řešit problémy, často ještě předtím, než mají šanci negativně ovlivnit obchodní operace. Například systémy mohou automaticky upravit alokaci zdrojů na základě analýzy využití, nebo mohou okamžitě izolovat a opravit zranitelné komponenty, čímž zvyšují celkovou bezpečnost IT prostředí. [34, 35]

Integrace monitoringu a automatizace v IT infrastruktuře vytváří robustní a dynamický systém, který může proaktivně reagovat na měnící se potřeby a výzvy. Tento přístup nejen že zlepšuje spolehlivost a výkon IT služeb, ale také poskytuje silný základ pro inovace. Automatizované monitorovací a správní procesy umožňují organizacím rychle se přizpůsobit novým technologiím a trendům, přičemž zajišťují, že jejich IT prostředí zůstává v bezpečí a v souladu s platnými předpisy. Výsledkem je IT ekosystém, který je nejen efektivní a bezpečný, ale také schopný podporovat dlouhodobý růst a inovace organizace.

## 3.5 IT prostředky v souvislosti s monitoringem

V této kapitole jsou teoreticky rozebrány ICT prvky, které budou implementovány a použity v souvislosti s monitoringem v praktické části práce. Opět by se jich dalo najít mnohem více, avšak zde budou rozebrány pouze nejvýznamnější prvky vzhledem k monitoringu.

### 3.5.1 Linux server

Servery, především OS Linux, jsou jednou ze zásadních částí této práce ať už v rámci implementace prostředků tak i jako samotný monitorovaný prvek. Proto je třeba si definovat jeho význam.

Co se historie samotného OS Linux týče, je tomu již přes 30 let, kdy Linus Torvalds v roce 1991 zveřejnil zprávu o novém OS, který navrhl. Torvalds původně tento systém vyvíjel jako hobby projekt s cílem vytvořit alternativní, volně dostupnou open-source verzi operačního systému MINIX, který byl založen na principech a designu Unixu. Za OS Linux

tedy stojí tři dekády obrovských pokroků a změn. Během těchto let si Linux upevnil svoji pozici jako jeden z nejvýznamnějších OS v oblasti ICT technologií. V současné době je Linux nejen základem pro mnoho serverových infrastruktur, ale také klíčovým prvkem v rozvoji cloudových technologií, IoT, mobilních zařízení a mnoha dalších inovativních aplikací. Význam Linuxu pro firmy spočívá v jeho flexibilitě, otevřenosti a bezpečnosti, díky čemuž se jedná o ideální platformu pro vývoj a nasazení různých aplikací. [36, 37]

Pro tuto práci je však ze všech variant provedení OS Linux nejzásadnější právě Linux server. Linux server je tedy serverový OS založený na již zmíněném kernelu (jádru) Linuxu, který je open-source. To znamená, že uživatelé mají přístup k jeho zdrojovému kódu a mohou jej upravovat a distribuovat v souladu s licencí GNU – GPL. Linux server poskytuje podnikům možnost nízkonákladového doručování obsahu, aplikací a služeb svým klientům. [38]

Jednou z klíčových výhod Linux serveru je jeho vysoký výkon vzhledem k jeho malé zátěži HW, kde je server hostovaný. Dále díky absolutní míře možnosti customizace také umožňuje zkušeným správcům těchto serverů nastavení velice kvalitní bezpečnosti. Linuxové servery bezpochyby také nabízejí vysokou míru stability a škálovatelnosti. Tyto servery jsou postaveny na technologiích, které podporují schopnosti vytváření tzv. snapshotů, což také umožňuje například efektivní vytváření záloh. Linuxové servery také bývají zpravidla kompatibilní s cloudovými technologiemi, což usnadňuje škálování podnikového prostředí. Linux server může sloužit jako základ pro téměř jakékoli typy IT technik, včetně containerů nebo cloud-native aplikací. Vysoká flexibilita a široká podpora komunity činí z Linuxu ideální platformu pro servery a podnikové aplikace, což potvrzuje jeho význam v současných ICT technologiích. [37, 39]

#### 3.5.1.1 Distribuce

Vzhledem k tomu, že Linux je jak již bylo zmíněno, zdarma a open-source, mnoho vývojářů je povzbuzováno k úpravě kódu a vytváření vlastních verzí. Díky tomu existují tisíce různých verzí nebo tzv. distribucí Linuxu a samotného Linux serveru. Některé verze se staly mimořádně populárními a vyvinuly silné základny uživatelů a přispěvatelů. Mezi nejpopulárnější distribuce patří Ubuntu, Debian, Redhat, Mint, nebo třeba Fedora. Pro tuto práci je však nejzásadnější právě distribuce Ubuntu, případně Debian. [40]

Ubuntu Server je extrémně výkonná distribuce Linuxu, která je speciálně navržena pro servery a síťová zařízení. Tato distribuce poskytuje všestranné řešení pro širokou škálu serverových aplikací – od výstavby rozsáhlých databázových systémů po provoz malých kancelářských souborových serverů. Vyznačuje se výjimečnou flexibilitou, díky které dokáže splnit a často i překonat specifické požadavky uživatelů. Ubuntu Server je tak ideální kombinací moderních vývojových technologií a osvědčené systémové stability. Díky komplexní podpoře HW lze Ubuntu Server instalovat nejen na starší HW infrastruktury, ale i na nejnovější serverové platformy. Tato široká kompatibilita zajišťuje, že Ubuntu Server může efektivně využívat výkon současného i budoucího serverového HW, což umožňuje organizacím udržet krok s technologickým vývojem bez nutnosti časté výměny operačních systémů. Ubuntu Server navíc nabízí bohatou kolekci předinstalovaných SW packages a snadno použitelných managerů těchto packages, které značně zjednodušují proces instalace a správy serverových aplikací. Jeho robustní bezpečnostní systém a pravidelné aktualizace zabezpečení chrání citlivá data a infrastrukturu před neustále se vyvíjejícími hrozbami. Všechny tyto vlastnosti činí Ubuntu server spolehlivou a udržitelnou platformou pro nejrůznější podnikové a síťové aplikace. [41]

Ubuntu server pravidelně vydává nové verze, což uživatelům nabízí přístup k nejnovějším funkcím, bezpečnostním opravám a aktualizacím. Mezi nejvýznamnější patří verze s dlouhodobou podporou tzv. LTS, které jsou ideální pro podnikové prostředí díky pětileté podpoře zabezpečení a aktualizací. Mezi příklady, jsou v této práci využity je možné zmínit například poslední verzi Ubuntu 22.04 LTS známou jako „Jammy Jellyfish“, která přinesla pokroky v cloudových technologiích a zabezpečení. Dále starší Ubuntu 20.04 LTS s přezdívkou „Focal Fossa“, zaměřená na vylepšení v oblasti výkonu a podpory HW, Ubuntu 18.04 LTS, známá jako „Bionic Beaver“, kde bylo přidáno vylepšení pro containerové technologie a automatizaci, a také Ubuntu 16.04 LTS, „Xenial Xerus“, jež se soustředí na stabilitu a kompatibilitu s různými aplikacemi. Tyto verze společně reflektují závazek Ubuntu Serveru k poskytování robustních a spolehlivých systémů s nejnovějšími technologickými inovacemi. [42]

Oproti tomu distribuce Debian je jednou z nejstarších a nejuznávanějších distribucí Linuxu, která slouží jako základ pro mnoho dalších distribucí, včetně již zmíněného Ubuntu serveru. Debian je známý svou stabilitou, bezpečností a přísnými kritérii pro zařazení SW do svého oficiálního repositáře, což z něj činí ideální volbu pro servery, na kterých je kladen

důraz na dlouhodobou spolehlivost a bezpečný provoz. Na rozdíl od Ubuntu, který obsahuje veliké množství verzí, u kterých je často vyžadován jejich upgrade. Oproti tomu Debian preferuje starší, ale důkladně testované a stabilní verze, aby zajistil maximální kompatibilitu a minimalizoval potenciální rizika. Toto zaměření činí Debian ideální pro kritické podnikové aplikace, kde je stabilita a spolehlivost prioritou nad přístupem k nejnovějším funkcím. Zatímco tedy Ubuntu server může být vhodnější pro ty, kdo hledají nejnovější SW inovace a rozšířenou podporu pro nový HW, Debian zůstává volbou pro ty, kdo potřebují rock-solid platformu s dlouhodobou podporou. [43, 44]

#### 3.5.1.2 VPS

Dále je dobré pro tuto práci zmínit možné dělení serverů na virtuální (VPS) a fyzické, které jsou popsány v kapitole níže. VPS představují formu virtualizace serverů, která umožňuje uživatelům získat výhody dedikovaného serverového prostředí na sdílené fyzické HW infrastruktuře většinou od poskytovatele těchto služeb. Takový server je efektivně oddělen od ostatních virtuálních serverů na stejném fyzickém serveru, čímž zajišťuje izolaci, vlastní systémové zdroje, jako jsou CPU, RAM a disk, a možnost plného přizpůsobení implementovaného OS včetně nasazených aplikací. Tato flexibilita a izolace činí VPS ideálním řešením především pro středně velké aplikace, webové stránky s mírně vyšším provozem nebo pro vývojové a testovací prostředí, kde je potřeba simulovat chování dedikovaných serverů za zlomek ceny. [45]

Výhody VPS spočívají ve vyšší kontrole nad konfigurací serveru, lepší izolaci zdrojů ve srovnání se sdíleným hostingem, a možnosti škálování zdrojů podle aktuálních potřeb. Uživatelé mohou snadno přidávat nebo ubírat zdroje v závislosti na výkonnostních požadavcích jejich aplikací, což je činí vhodnými pro projekty, které očekávají růst. Oproti fyzickým serverům, VPS nabízejí snadnější správu a nižší náklady na HW. Avšak jsou zde i určitá úskalí. VPS mohou trpět omezeným přístupem k HW a potenciálně nižším výkonem z důvodu sdílení fyzických zdrojů. [45, 46]

#### 3.5.1.3 Fyzický server

Oproti tomu klasické fyzické servery, často označované jako dedikované servery, jsou samostatné HW jednotky běžně přidělené specificky dle daných potřeb aplikací nebo služeb. Tyto servery poskytují plnou výpočetní kapacitu jako CPU, RAM, disk a síťové

zdroje bez nutnosti sdílení s jinými uživateli nebo aplikacemi, což zaručuje maximální výkon a bezpečnost. Dedikované servery jsou ideální pro náročné aplikace, velké databáze, rozsáhlé webové stránky a podnikové aplikace, kde stabilita, výkon a kontrola nad fyzickým HW jsou klíčové. [45]

Hlavní výhody fyzických serverů jsou především nekompromisní výkon, vysoká úroveň bezpečnosti a exkluzivní přístup k HW zdrojům. Uživatelé mají možnost plně přizpůsobit HW konfiguraci a OS podle specifických potřeb, což z nich činí vhodné řešení pro specializované a výkonově náročné úlohy. Na druhou stranu, dedikované servery přinášejí vyšší počáteční a provozní náklady ve srovnání s VPS, a vyžadují pokročilejší technické znalosti pro jejich správu a údržbu. Na rozdíl od VPS, kde zdroje mohou být dynamicky škálovány, fyzické servery vyžadují fyzické upgrade pro rozšíření kapacity, což může být časově i finančně náročnější. [45, 47]

#### 3.5.1.4 Bash Scripty

Bash, oficiálně známý jako GNU Bourne Again Shell, je standardní příkazová řádka jako široce používaný shell a scriptovací jazyk pro většinu Linuxových distribucí a některé systémy UNIX. Jeho vývoj započal Brian Fox pod hlavičkou Free Software Foundation v roce 1989 a dnes je udržován Chetem Rameyem. Díky své příslušnosti k projektu GNU pro svobodný operační systém se Bash stal klíčovým nástrojem pro interaktivní práci v příkazové řádce, zpracování dávek příkazů a spouštění právě jednotlivých příkazů. [48]

Bash scripty jsou pak textové soubory obsahující sled příkazů pro Bash. Tyto tzv. scripty umožňují automatizovat rutinní a opakované příkazy do automatizovaných úloh. Mohou zahrnovat vše od systémové správy, jako je aktualizace SW a zálohování dat, po komplexnější úlohy, jako je zpracování a analýza dat, správa síťových operací nebo právě co se týká této práce, vytváření monitorovacích scriptů. Díky schopnosti Bash scriptů volat a integrovat jiné programy a služby, poskytují uživatelům mimořádnou flexibilitu a efektivitu při automatizaci širokého spektra úloh. [48]

Jedním z klíčových aspektů Bash je jeho výkonný scriptovací jazyk, který zahrnuje podporu pro obecné programovací konstrukce, jako jsou proměnné, kontrolní struktury a cykly. Tyto vlastnosti umožňují Bash scriptům vykonávat složité logické operace a manipulovat s daty s vysokou úrovní sofistikovanosti. Navíc Bash integruje řadu nástrojů

a programů UNIX, což umožňuje scriptům efektivně pracovat s textovými soubory, zpracovávat vstupy a výstupy a spravovat systémové procesy. [48, 49]

Scriptování v Bash nabízí tedy řadu výhod, včetně možnosti rychle a s minimálním úsilím automatizovat složité nebo časově náročné úlohy. Scripty jsou snadno přenositelné mezi různými systémy, což umožňuje uživatelům a správcům systémů DevOps využívat osvědčené postupy a rychle implementovat daná řešení. Kromě toho scriptovací jazyk Bash a jeho prostředí podporují rychlý vývoj a testování, což uživatelům umožňuje experimentovat a iterativně zdokonalovat své scripty s okamžitou zpětnou vazbou. [50]

Přestože scriptování v Bash přináší mnoho výhod, existují také určitá omezení a výzvy. Komplexní scripty mohou být obtížně čitelné pro ty, kteří nejsou obeznámeni se syntaxí shellu, a výkon scriptů může být v některých případech omezen sdílením prostředků s ostatními procesy na stejném systému. Navíc, ačkoliv Bash poskytuje silné nástroje pro automatizaci a správu systémů, vyžaduje dobrou znalost příkazové řádky a programovacích konceptů pro efektivní využití. [50]

#### 3.5.1.5 Cron

Cron je služba defaultně nasazena v OS Linux, která umožňuje správcům systémů DevOps plánovat spouštění scriptů a příkazů v předem určených časech. Tato služba je tedy zásadní pro automatizaci úloh, které musí být prováděny pravidelně, jako jsou například běžné rutinní úlohy typu zálohování dat, aktualizace SW, nebo i složitější jako právě v této práci vytvářené monitorovací scripty. Cron využívá speciální konfigurační soubor, známý jako crontab, pro uchování plánů úloh. Každý uživatel systému může mít svůj vlastní crontab, a existuje také systémový crontab pro úlohy, které se týkají celého systému. [51]

Použití Cronu zahrnuje definici přesného času a data, kdy má být úloha spuštěna, spolu s příkazem nebo scriptem, který má být vykonán. Formát pro zadání úloh je flexibilní, umožňuje specifikovat spouštění v minutových, hodinových, denních, týdenních, měsíčních intervalech, nebo kombinaci těchto intervalů. Tyto konkrétní intervaly se definují pomocí pěti pozic v defaultním stavu označovaných jako * * * * *, kdy každá z pozic značí jeden z typů intervalů, které jsou možné nadefinovat číselnými hodnotami, případně určitými znaky. Díky této flexibilitě mohou uživatelé přesně naplánovat, kdy se mají provádět různé předdefinované operace. [52]

Význam Cronu v Linuxu tedy spočívá v jeho schopnosti efektivně automatizovat opakující se úlohy, což umožňuje uživatelům a správcům systémů udržovat systémy bezpečné, efektivní a v optimálním stavu s minimálním úsilím. Jeho použití pro spouštění monitorovacích scriptů je příkladem, jak může Cron přispět k proaktivní správě IT infrastruktury a zajištění její spolehlivosti a výkonu. Toto využití Cronu nabízí několik výhod, včetně možnosti předem naplánovat úlohy pro optimální časy. Dále minimalizuje rizika lidských chyb tím, že se odstraní manuální repetitivní spouštění úloh zaměstnanci.

#### 3.5.1.6 Nginx

Nginx je výkonný a efektivní webový server a zároveň reverzní proxy server, který se vyznačuje svou vysokou dostupností tzv. high availability, škálovatelností a nízkou náročností vzhledem k RAM. Jeho schopnost efektivně zpracovávat statický obsah a zároveň fungovat jako reverzní proxy server pro moderní webové aplikace (například v prostředích Node.js, PHP a dalších v případě této práce s nasazením v Docker containerech) ho činí ideální volbou pro mnohé internetové projekty. V kombinaci s Apache httpd serverem může Nginx vylepšit rychlost obsluhy požadavků na statické soubory a celkově zlepšit výkon webu, jak z pohledu klienta, tak serveru. V kontextu této kapitoly je důležité zmínit i souvislost Nginx s DNS (Domain Name System) a SSL (Secure Sockets Layer). [53, 54]

DNS je zásadní pro fungování samotných aplikací na internetu, protože překládá snadno zapamatovatelná doménová jména na IP adresy, které jsou používány pro navázání komunikace se servery. Tato databáze je distribuována po celém světě a obsahuje záznamy, které definují vztah mezi názvem domén a jejich IP adresy. Každé IP může mít přiřazeno několik doménových jmen, což usnadňuje orientaci v adresách internetových stránek. [55]

SSL je zase protokol navržený k zajištění bezpečné komunikace mezi servery a jejich klienty na internetu. Funguje tak, že data ještě před odesláním šifruje a po přijetí na druhé straně dešifruje, čímž zajistí jejich důvěrnost a integritu. Nginx podporuje SSL velice efektivně, což umožňuje webům používat protokoly https, bezpečnější variantu protokolů http, k ochraně přenášených dat. Nginx je široce využíván právě pro svou podporu SSL a jeho schopnost snadno konfigurovat SSL certifikáty pro zabezpečení komunikace. V rámci implementace v této práci jsou SSL certifikáty konfigurovány pomocí nástroje přímo od Nginx s názvem Certbot. [56]

V této práci je Nginx nástrojem pro implementaci nasazovaných monitorovacích nástrojů, včetně se zmíněnými DNS a SSL, které jsou zde zároveň předmětem reprezentující prostředky, které jsou monitorovány.

### 3.5.2 VPN

Virtuální privátní sítě neboli VPN jsou technologií navrženou k vytvoření bezpečného a šifrovaného komunikačního kanálu přes veřejné nebo nezabezpečené sítě, jako je internet. Tento princip umožňuje uživatelům a organizacím bezpečně přistupovat k vzdáleným síťovým zdrojům nebo internetu, jako by byli připojeni přímo k privátní síti. Tato funkčnost je zvláště užitečná pro podniková prostředí, kde zaměstnanci potřebují bezpečný přístup k firemním datům z různých geografických umístění, a to i při práci z domova nebo na cestách. VPN také hrají klíčovou roli v ochraně soukromí uživatelů tím, že maskují jejich skutečné IP adresy a šifrují data, čímž brání odposlouchávání a sledování jejich internetové aktivity. Navzdory své historické reputaci jako technologie náročná na používání se moderní řešení VPN, díky pokroku v oblasti uživatelské přívětivosti a bezpečnosti, stala snadno dostupnou a široce využívanou pro různé účely. V této práci jsou VPN certifikáty nejen prostředkem pro bezpečnou implementaci. Některé servery však také mají přístup omezen skrze firewall UFW pouze z adres těchto VPN. Ale jsou zde také jako prostředky, které se monitorují. [57]

V poslední době však vznik nových technologií a protokolů výrazně zjednodušil a zefektivnil implementaci a následné používání VPN certifikátů. Těmito technologiemi jsou například OpenVPN nebo WireGuard, které jsou ve vlastní práci níže probírány. Tyto nástroje nabízejí lepší bezpečnost, vyšší rychlost a snazší konfiguraci, což z VPN činí přívětivější a přístupnější řešení pro širokou škálu uživatelů. OpenVPN je široce používaný, open-source SW, který podporuje různé metody šifrování, zatímco WireGuard představuje novější přístup s důrazem na jednoduchost a vysokou výkonnost.

### 3.5.3 Containerizace

Containerizace představuje moderní a v dnešní době velice často využívanou metodu pro správu a nasazování aplikací. Přináší revoluční přístup k virtualizaci. Tato technologie umožňuje „balení“ aplikací do izolovaných a přenositelných balíčků celého prostředí aplikací tzv. containerů, které obsahují všechny potřebné závislosti a konfigurace. Přestože

tyto containery využívají zdroje hostitelského OS jako jsou oddíly CPU, RAM atp., fungují izolovaně od ostatních procesů probíhajících na daném hostitelském OS. Containerizace nabízí významné výhody v efektivitě, rychlosti a snížení nákladů na infrastrukturu. Oproti tradičním virtuálním strojům, které vyžadují kompletní OS a hypervizory pro abstrakci HW, containery zjednodušují nasazování a správu aplikací a microservices díky rychlému startu a menšímu nároku na zdroje. Vývojové týmy tak mohou snadněji a rychleji reagovat na potřeby trhu a zákazníků, což z containerizace činí klíčový prvek v moderních DevOps a cloudových architekturách. [17, 58, 59]

Oproti tradičním virtuálním strojům, které simuluji celý OS a vyžadují hypervisor pro abstrakci HW, containery tedy přinášejí zjednodušení tím, že eliminují potřebu dalšího OS pro každý virtuální stroj. V praxi je možné se běžně setkat s několika nasazenými containery na jednom hostitelském OS, klidně v řádu desítek. Toto uspořádání umožňuje containerům být výrazně šetrnější vzhledem k zátěži a rychlejší při spouštění než jejich protějšky v podobě virtuálních strojů. Navíc díky sdílení hostitelského OS, containery efektivně využívají systémové zdroje, což vede k rychlejšímu škálování a distribuci aplikací. [59]

Historie a vývoj containerizace sahají do 60. let 20. století, kdy byla virtualizace využívána pro oddělení uživatelů na hlavních počítačových systémech. V moderní době, především díky vývoji technologií jako Linux VServer, Solaris Containers, a později LXC (LinuX Containers), se containerizace stala klíčovou součástí DevOps praxí, poskytující rychlý vývoj, testování a nasazování aplikací. [17, 58]

Izolace containerů a jejich simultálního spouštění se dosahuje pomocí nízko úrovňových mechanismů kernelu, jako jsou Linux namespaces a cgroups. Namespaces zajišťují izolaci systémových zdrojů, takže procesy běžící v jednom containeru nemohou vidět procesy běžící v jiných containerech nebo na hostitelském OS. Cgroups (Control Groups) pak omezuje, upřednostňuje a alokuje fyzické zdroje, jako je CPU, RAM, a I/O kapacita, pro nasazené containery. Tento přístup umožňuje containerům sdílet jádro hostitelského OS, ale přitom být absolutně izolované, což vede k vysoké úrovni efektivity a snížení režijních nákladů na zdroje. Technologie containerizace tak přináší podstatné zvýšení hustoty nasazení aplikací a umožňuje rychlé škálování aplikací podle aktuálních potřeb, což je zásadní pro dynamické prostředí moderního SW vývoje a provozu. [17, 58]

#### 3.5.3.1 Docker

Již zmíněná technologie efektivní containerizace Docker je platforma a produkt společnosti Docker Inc. Tato společnost byla založena Solomonem Hykesem. Zpočátku byl Docker vyvinut jako nástroj v rámci platformy dotCloud. Avšak Docker se rychle stal jednou z nejpoužívanějších technologií celé myšlenky containerizace, umožňující vývojářům snadno vytvářet a spravovat containery jak na Linuxu, tak na Windows. Docker, vycházející z otevřeného projektu Moby, se skládá z několika komponent jako containerd a runc. Tyto komponenty pracující na vytvoření a správě containerů. Docker je licencován pod otevřenou licencí Apache 2.0. [17, 60]

Základem Dockeru je Docker Engine, pracující na principu klient-server aplikace. Engine se skládá z daemona (na straně serveru), REST API a CLI (na straně klienta). Tyto prvky společně umožňují správu Docker objektů, jako jsou containery, image, network a volumes. Komunikace mezi klientem a daemonem probíhá pomocí REST API, což umožňuje jak interakci přes příkazovou řádku, tak pomocí určitých scriptů. [60]

Jedním z klíčových souborů pro práci s Dockerem je *Dockerfile*. *Dockerfile* je textový dokument obsahující všechny příkazy potřebné k automatickému sestavení (tzv. buildu) Docker image. Image představuje publikovaný odkaz na statické soubory obsahující vše potřebné pro spuštění aplikace v containeru. Tyto image se poté spouštějí jako containery, což jsou spustitelné instance daných imagů běžící izolovaně. Dalším zásadním souborem ve vztahu k dockeru je *docker-compose.yml.* Tento soubor umožňuje definici a spuštění více containerových Docker aplikací, kde každá služba v aplikaci může být definována jako samostatný container. To umožňuje efektivní a snadné nasazení těchto containerů s možností přehledného uchování konfigurace. [59, 61]

Docker nabízí dvě edice enginu, kterým je Community Edition (CE) pro individuální vývojáře nebo malé týmy začínající s Dockerem, a Enterprise Edition (EE) pro podnikové aplikace s pokročilými bezpečnostními funkcemi a zárukami služeb. Docker Hub a Docker Registry pak představují klíčové komponenty ekosystému Dockeru, poskytující repositáře pro ukládání, sdílení a distribuci Docker image. [60]

Ve světle této práce jsou Docker containery s využitím Docker CE enginu používány nejen jako technologie pro nasazení všech implementovaných nástrojů a prostředků, ale zároveň představují objekt monitorování, což ukazuje na jejich flexibilitu a možnou

integraci do širších systémů IT infrastruktury. Na rozdíl od Kubernetes, který je zaměřen na orchestraci containerů ve velkém měřítku, Docker nabízí jednodušší, ale stále robustní řešení pro správu containerů. Tím se stává ideálním výchozím bodem pro tuto práci.

## 3.6 Analýza dostupných monitorovacích nástrojů

V této kapitole jsou provedeny slovní analýzy k nástrojům, které je možné využít pro monitoring IT prostředků. Na základě této analýzy budou následně některé z těchto nástrojů vybrány pro implementaci v podniku níže ve vlastní práci. Analýzy jsou zpracovávány jak na základě odborné literatury a sekundárních zdrojů, tak i na základě zkušeností z praxe.

### 3.6.1 Grafana

Prvním zmíněným nástrojem je Grafana jakožto výkonná platforma pro vizualizaci a analýzu dat. Grafana se stala nedílnou součástí mnoha monitorovacích strategií v celém spektru především průmyslových odvětví. Díky své schopnosti integrovat široké spektrem datových zdrojů, od relačních databází po time series data a NoSQL databáze, soubory Excel nebo CSV, až po metriky získané pomocí jiných monitorovacích nástrojů. Grafana tak poskytuje unikátní flexibilitu v přístupu k datům. Tato platforma umožňuje snadno vytvářet detailní a interaktivní dashboardy, které nabízejí hluboký vhled do výkonu aplikací, infrastruktury a mnoha dalších oblastí. V rámci dashboardů Grafana využívá pro vizualizaci jednotlivé části, tzv. panely, které mohou být každý nadefinován zcela odlišným způsobem. Tato široká škála možných typů vizualizace dat v panelech nabízí veliké množství graficky a funkčně odlišných vizualizací, od časových řad, přes statistické grafy, až například po velice specifické geografické mapy. Jedním z klíčových prvků Grafany její schopnost transformace dat přímo v panelu vizualizace, což umožňuje uživatelům provádět všelijaké matematické výpočty, logické operace, přejmenování, nadefinování vizuální podoby a mnoho dalších. To vše je možné provádět s minimálním úsilím a bez nutnosti pokročilých dotazů nebo manipulace s daty ve zdrojovém systému. Tato funkcionalita, společně s centralizovanou správou alertů, usnadňuje správu upozornění a zlepšuje reakční schopnosti na potenciální problémy v systémech. [62]

Grafanu je také možné nasadit prostřednictvím Dockeru, což výrazně usnadňuje a urychluje samotné nasazení, včetně konfigurace a procesu zavádění monitorovacích řešení. Nasazení Grafany jako Docker containeru také velice usnadňuje škálovatelnost. Další

výraznou výhodou nasazení v Dockeru je fakt, že Grafana se v této formě skládá pouze z jednoho containeru, který zároveň nepředstavuje výraznou zátěž na prostředky OS. Využití Grafany pro nasazení v containerech Docker zahrnuje dvě klíčové metody: přímou práci s Docker CLI a použití *docker-compose.yml* pro efektivní správu a zálohu konfigurace tohoto containeru. [65]

Grafanu je zároveň možné v rámci této práce použít jako nástroj pro vizualizaci všech implementovaných řešení, nehledě na odlišnost jejich sběru, uchovávání a publikování dat směrem ke Grafaně. Výjimkou mohou být řešení, která mají vlastní vizualizační prostředí. Je tedy možné integrovat veliké množství prakticky všech datových zdrojů, tzv. data sources. Tyto odlišné datové zdroje je zároveň možné kombinovat nejen v rámci jednoho dashboardu, ale také v rámci jednoho konkrétního panelu za využití multi query nadefinování daného panelu. Díky této rozsáhlé podpoře pro různé datové zdroje a jejich integraci s monitorovacími a alertovacími systémy je Grafana ideálním kandidátem pro centralizovanou vizualizační vrstvu v rámci komplexních IT prostředí. [63, 64]

Další zásadní již zmíněnou funkcí Grafany je možnost alertingu. Ten je zásadním podnástrojem Grafany pro možnost okamžitého všimnutí a rychlou reakci na vzniklé nebo blížící se incidenty. Tento alerting umožňuje definovat komplexní pravidla upozornění přímo v panelech dashboardů. Tato pravidla mohou spouštět notifikace přes různé kanály, včetně MS Teams, Slacku nebo třeba e-mailu. Tyto notifikace se odesílají kdykoli, pokud jsou splněny specificky nadefinované podmínky. Těmito podmínkami může být například překročení limitních hodnot využití zdrojů, nebo třeba jen přiblížení se k těmto hodnotám pro možnost včasné reakce. Tato kombinace pokročilých alertingových funkcí a intuitivního UI činí Grafanu ideální volbou pro proaktivní monitorování a analýzu výkonnosti systémů v reálném čase. [62]

V kontextu této práce, kde je kladen důraz na přehlednost, efektivitu a adaptabilitu monitorovacích řešení, Grafana exceluje díky své schopnosti přizpůsobit se různým zdrojům dat a poskytnout nástroj pro hloubkovou analýzu a přehled o stavu monitorovaných systémů. Grafana nabízí také robustní podporu pro alerting a notifikace. Spolu s její schopností integrace s širokou škálou datových zdrojů a externích služeb, ji činí nepostradatelnou součástí efektivní strategie monitorování. Navíc vzhledem k nízkému zatížení zdrojů OS při možném nasazení jako Docker container je tak Grafana doporučena

v rámci této práce jako výhradní vizualizační nástroj pro všechny use cases a jejich řešení, která nemají vlastní vizualizační UI.

### 3.6.2 Prometheus

Dalším probíraným nástrojem je Prometheus, jakožto přední nástroj pro monitorování, ukládání získaných dat a vytváření alertingu v moderních cloudových a containerizovaných prostředích. Jako projekt, který je součástí Cloud Native Computing Foundation (CNCF), představuje Prometheus robustní a flexibilní nástroj, který splňuje požadavky dynamických prostředí současného SW inženýrství. Jeho schopnost shromažďování dat typu časových řad z různých zdrojů a poskytování propracovaného jazyka PromQL pro možné nadefinování query při vizualizaci hodnot umožňuje efektivně analyzovat a vyhodnocovat výkonnost a využití sledovaných prostředků. Díky své architektuře založené na „pull“ modelu Prometheus aktivně získává metriky z definovaných cílů. Je také vhodný pro monitorování ve velkém měřítku, aniž by bylo nutné konfigurovat pro sběr dat složité agenty. Namísto agentů je možné využít širokou škálu integrovaných exporterů. Pomocí nich je možné Prometheus snadno rozšířit o monitorování velikého množství různých systémů, aplikací a jejich prostředků. [66, 68]

Základem nástroje Prometheus je tedy jeho schopnost shromažďovat data typu časových řad prostřednictvím modelu, kdy Prometheus periodicky získává metriky z koncových monitorovaných bodů, kde je nasazen některý z exporterů dat. Tento „pull“ model umožňuje efektivní sběr dat, který minimalizuje zátěž na monitorované systémy. Zároveň je možné tento model snadno přizpůsobit dle specifických požadavků. Tato získaná data je následně možné pomocí již zmíněného jazyka PromQL efektivně nadefinovat pro vizualizaci. Tento jazyk umožňuje vytvářet všelijaké matematické výpočty a logické operace za použití jedné nebo více získaných metrik pro maximální možnost přizpůsobení vytvářené vizualizace. [66, 67]

Další z klíčových funkcí nástroje Prometheus je tak jako u Grafany i zde systém alertingu. Ten je zajištěn prostřednictvím Alertmanageru. Alertmanager umožňuje definovat pokročilá pravidla pro generování alertů na základě analyzovaných dat. Tato flexibilita a centralizovaná správa upozornění posilují schopnost rychle reagovat na potenciální problémy a zajišťují vysokou dostupnost monitorovaných služeb. V případě této práce by však nebylo potřebné využívat alerting ze dvou nástrojů. [66]

Prometheus není tedy omezen pouze na sběr dat přímo z aplikací, které nativně podporují jeho formát expozice metrik. Díky široké škále dostupných exporterů od různých tvůrců je možné Prometheus v rámci sběru dat snadno integrovat s různými systémy, službami a jejich prostředky, které Prometheus nemá defaultně přednastavené. Exportery, jako jsou node-exporter a container-exporter, které jsou pro tuto práci doporučeny, rozšiřují možnosti monitorování o sběr metrik specifických pro OS Linux a Docker containery. Tyto nástroje usnadňují získávání komplexního přehledu o stavu a výkonu infrastruktury úprav monitorovaných nástrojů. [69]

Vzhledem k tomu, že přístupnost nástroje Prometheus umožňuje integrovat získaná data z exporterů do vizualizačního nástroje Grafana, se nástroj jeví jako vhodný pro použití v této práci. Dalšími výhodami je tedy přehlednost tohoto nástroje včetně metrik, které je možno získávat pomocí využití několika různých variant exporterů. Současně je možné exportery a samotný nástroj Prometheus nasadit jako Docker container s poměrně jednoduchou konfigurací. Každý z nástrojů také nevyžaduje více než jeden container a nejsou tak výrazným zatížením serverů. Na základě všech těchto informací je nástroj Prometheus také doporučen pro následnou implementaci ve vlastní práci.

### 3.6.3 Influx DB

Předchozí zmíněný nástroj pokryje velikou část možných systémů, aplikací a jejich prvků, které lze monitorovat, avšak jsou zde stále případy, kde je potřeba tak specifický monitoring, že ani klasické exportery v případě využití nástroje Prometheus nedostačují. Z tohoto důvodu je potřeba přidání vlastních specifických úloh pro získávání dat specifických řešení. Zde by však bylo příliš složité vytváření kompletních exporterů pro integraci do nástroje Prometheus. Z toho důvodu budou pro sběr dat těchto specifických požadavků použity Bash scripty přímo v Linuxovém prostředí Bash. Avšak tu vzniká potřeba nástroje pro možné zasílání těchto dat, jejich uchovávání a následné předávání vizualizačnímu nástroji Grafana.

InfluxDB je přední časově orientovaná (tzv. time series database TSDB) databáze navržená speciálně pro ukládání a správu dat v čase. Díky tomu je možné efektivně monitorovat nejen systémy, aplikace, IoT zařízení a provádět analýzy v reálném čase. Tato specializovaná databáze umožňuje snadno ukládat, spravovat a analyzovat data, která jsou podmínečně spojena s označením času, tzv. timestamps. To bývá typickou charakteristikou

dat generovaných IoT zařízeními a aplikacemi. InfluxDB se zároveň vyznačuje výkonem a efektivitou, což umožňuje rychlou analýzu a vizualizaci dat v reálném čase. [62]

Proces instalace a konfigurace InfluxDB je navržen tak, aby byl co nejjednodušší, s podporou pro různé platformy včetně 64bitových systémů jako nejen distribuce Linux serveru Ubuntu, zásadní pro tuto práci, ale také systémy typické pro IoT řešení, jako například Raspberry Pi. Díky možnosti nasazení InfluxDB ve formě Docker containeru je jeho škálování, správa a nasazení značně usnadněno. Zároveň se tento nástroj skládá pouze z jednoho containeru a není nijak náročný na hostitelský OS. Dokonce i v případě nasazení nástroje v Dockeru umožňuje dokončení konfigurace pouze v samotném UI, což z InfluxDB činí ideální řešení pro různé aplikace, od malých projektů až po projekty velice zatížené daty. [62]

Schopnost zpracovávat velké objemy dat s vysokou přesností a minimální latencí je jednou z významných funkcí InfluxDB. Databáze používá víceúrovňové ukládání dat k optimalizaci výkonu dotazů a umožňuje jejich kompresi s vysokým poměrem, což vede k výraznému snížení nákladů na úložiště. Podpora pro SQL a podobné dotazovací jazyky usnadňuje integraci a rozvoj aplikací. V samotném UI nástroje je pak možné vytvářet odlišné organizace a buckety pro různá použití. Koncept bucketů v InfluxDB je podobný konceptu databází v tradičních relačních databázových systémech. Každý bucket v InfluxDB obsahuje sadu time series dat, které jsou seskupeny dohromady na základě společného účelu nebo tématu. Každý bucket má definovanou dobu uchování (tzv. retention policy), která určuje, jak dlouho mají být data v bucketu uchovávána před jejich automatickým odstraněním, aby se spravovalo využití úložiště a udržovala efektivita databáze. V případě potřeby je tedy možné tento nástroj velice dobře přizpůsobit dle různých specifických požadavků. [70, 71]

Pro integraci a sběr dat z různých zdrojů, včetně bash scriptů důležitých pro tuto práci, je možné použít curl requesty typu POST. Mezi sebou se tyto requesty odlišují uvedením konkrétní organizace a bucketu. Pro zabezpečení zasílaných dat jsou zde také použity API tokeny generované samotnou databází. Tyto requesty však musí mít jasnou strukturu. Ta obsahuje přesnou syntaxi názvu metriky, jejich proměnných, hodnot a také již zmíněné timestampy. [71]

Na základě těchto informací je tedy InfluxDB doporučen pro specifická řešení, kde je využití exporterů s integrací do nástroje Prometheus nedostačující. Přestože InfluxDB nabízí vlastní možnosti vizualizace, tak vzhledem ke sjednocení vizualizace monitoringu

v této práci a analýze dat bude InfluxDB integrován také do Grafany. To umožní vytvářet komplexní a přehledné dashboardy. S přihlédnutím k tomu, že s nasazením v Dockeru je konfigurace tohoto nástroje velice jednoduchá a container samotný není náročný na systém, jeví se použití nástroje InfluxDB jako dobré řešení. [70, 71]

### 3.6.4 Loki

Předchozí analyzované nástroje však neumožňují monitoring záznamů samotných aplikací, tzv. logů. Pro tento účel je vybrán a analyzován nástroj Loki. Jedná se o moderní systém pro shromažďování a správu logů, vyvinutý společností Grafana Labs. Jeho hlavním cílem je poskytnout efektivní řešení pro shromažďování, ukládání a vyhledávání logů v cloudových a microservices architekturách. Loki se zaměřuje na jednoduchost a efektivitu, což ho odlišuje od jiných nástrojů pro správu logů, které jsou často zatíženy složitostí a vysokými náklady na ukládání. [72]

Loki je inspirován nástrojem Prometheus. Avšak po technické stránce oproti monitorování a ukládání time series dat v případě nástroje Prometheus, Loki indexuje metadata logových zpráv a obsah těchto logů ukládá odděleně. Tato architektura umožňuje, aby vyhledávání informací v logách bylo rychlé a efektivní, zatímco jejich samotný obsah může být uchováván na levnějších typech úložišť. To je pro monitoring logů velice důležité vzhledem k často velikému objemu dat od určitých aplikací, které mohou logovat i desítky řádků za minutu. Tento sběr logových dat může být zajištěn více způsoby. První je použití nástroje pro export logů Promtail také od Grafana Labs, přímo související s nástrojem Loki. Tato varianta vytváří soubory s logy, tzv. logfily a ty následně zasílá nástroji Loki. Loki však není uzavřen pouze pro tuto svou architekturu a umožňuje také použití vlastních řešení pro vytahování a odesílání logů, jako například JSON scripty. [73, 74]

Tím, že je Loki vyvinut přímo společností Grafana Labs, je jasné, že je možnost tato získaná data velice dobře integrovat do vizualizačního nástroje Grafana. Pro následnou konfiguraci panelů obsahující data z nástroje Loki je možné použít pro nadefinování qery propracovaný jazyk LogQL. V tomto případě je další výhodou efektivita a nízké náklady na úložiště, což jak bylo zmíněno je u logů velice důležité. Tento nástroj včetně jeho exporterů je také možno velice snadno nasadit jako Docker container. Jak u nástroje Loki, tak i u Promtailu, je sice potřeba dodatečná konfigurace, která však nepředstavuje žádný složitý

proces. Opět i v tomto případě se jednotlivé nástroje skládají pouze z jednoho containeru, což opět šetří zdroje hostitelského OS. [74]

Na základě těchto informací je nástroj Loki, včetně jeho exporteru Promtail, také doporučen pro následnou implementaci ve vlastní práci. Toto řešení je možné použít u případů, kde je potřebné vývojářům vystavit logy jejich aplikací pro možné debuggování.

### 3.6.5 Portainer

Dalším analyzovaným nástrojem je Portainer. Ten je určen pro správu, monitoring a orchestraci Docker containerů přes přívětivé UI. Jeho hlavním cílem je poskytnout jednoduché, intuitivní a vizuálně přehledné prostředí pro správu containerů, bez nutnosti hluboké znalosti CLI příkazů Dockeru. Tato platforma se stala populární zejména mezi DevOps zaměstnanci s ohledem na možnost poskytnout vývojářům aplikací, které jsou nasazovány v Dockeru, detailní přehled o jejich stavu a možnost správy. Existuje několik možností využití. Od monitoringu stavu běžících Docker containerů a také jejich logů, až po možnost konfigurace síťového nastavení, spravování volumes a nastavování politiky přístupu. Pro zkušenější vývojáře se znalostí Docker příkazů je možné také spustit samotnou consoli Docker containeru v Portainer UI a provádět jakékoli změny bez nutnosti přístupu k serveru, kde je container nasazen. Díky vizuálně přehlednému UI je snadné identifikovat a řešit problémy, provádět aktualizace a optimalizovat prostředky. Portainer podporuje správu více Docker prostředí, což umožňuje centralizovanou správu containerů napříč různými servery a cloudovými platformami do jednoho nástroje s pouze jednou vizualizací. [75]

Na technické úrovni Portainer využívá architekturu založenou na agentech. Tato architektura umožňuje do Portaineru distribuovat pro možnou správu různá Docker prostředí najednou, což zjednodušuje monitoring, správu a nasazování aplikací v rozsáhlých a komplexních infrastrukturách. Agenti Portaineru jsou jednoduché služby, které se nasazují na servery, a ty hostují sledované Docker containery. Každý agent shromažďuje informace o stavu, metrikách a logách z containerů a Docker prostředí, na kterém je nasazen, a odesílá tyto informace zpět do centrálního Portainer serveru. Tento přístup založený na agentech také umožňují Portaineru provádět důkladnější monitoring a využívat pokročilé funkce správy, jako je například automatizované nasazování, škálování nebo aktualizace containerů.

Kromě toho systém agentů Portainer zajišťuje dodržování bezpečnostních politik a poskytuje izolovanou komunikační vrstvu, která minimalizuje riziko neoprávněného přístupu k Docker daemonu. Komunikace mezi Portainer serverem a agenty je zabezpečena pomocí TLS, což zajišťuje, že všechna data přenášená mezi nimi jsou šifrována a chráněna před odposlechem. Zároveň jako u všech předešlých nástrojů je možné i tento efektivně nasadit jako samostatný Docker container s možnou konfigurací pouze v UI nástroje po nasazení. To samé se týká samotných agentů, kde ani žádná dodatečná konfigurace není třeba. To představuje šetrné řešení, jak vzhledem ke zdrojům serveru, na kterém je nasazen Portainer samotný, tak i k monitorovaným serverům, kde se nacházejí jeho agenti. Na základě toho je i toto řešení doporučeno pro implementaci ve vlastní práci, přestože se nejedná o čistě monitorovací nástroj. [76]

### 3.6.6 Nástroje, které nejsou doporučeny pro vlastní práci

Zbylé analyzované nástroje jsou přesunuty do této podkapitoly, neboť z určitých důvodů, které jsou ke každému z nástrojů uvedeny, nesplňují požadavky nebo jsou zkrátka nevyhovující pro implementaci v této práci. Těmito důvody může být například veliká zátěž na systém, složitá konfigurace, vysoké náklady na provoz, zaměření pouze na úzké spektrum monitorovaných prostředků, nekompatibilita s ostatními doporučenými nástroji, nadbytečnost a další.

#### 3.6.6.1 ELK

ELK je sjednocením několika nástrojů do jednoho, tzv. stack, kombinující Elasticsearch, Logstash a Kibanu. ELK je široce uznávaným řešením pro sběr, vyhledávání a vizualizaci dat a logů v reálném čase. Každá komponenta v tomto triu hraje specifickou roli. Logstash se stará o zpracování a předávání dat, Elasticsearch slouží jako vyhledávací a analytický engine a Kibana zprostředkovává vizualizaci získaných dat. Společně poskytují silný nástroj pro monitorování a analýzu logů a datových proudů. I přes svou popularitu a schopnosti existují určité aspekty a omezení ELK stacku, které mohou být pro některé implementace problematické. [77]

Nejzásadnějším problémem v souvislost s touto prací je vysoké využití zdrojů. Zejména Elasticsearch může být v rozsáhlých nasazeních velmi náročný na RAM a CPU, což může vést k přetížení infrastruktury. To může být pro malý podnik, jako je Sabo, který

je detailněji představen níže, překážkou zejména v případě, kdy jde o škálování firemní monitorovací infrastruktury. ELK je náročný i v případě nasazení v Dockeru, neboť se běžně tento stack skládá ze tří containerů. Dalším možným problémem, kterému je možné u ELK stacku čelit, je komplexní konfigurace a správa. Nastavení Logstash pipeline, optimalizace Elasticsearch clusteru a správa Kibany mohou vyžadovat hluboké znalosti a zkušenosti každého z jednotlivých nástrojů. Toto může být zvláště obtížné pro týmy s omezenými zdroji nebo pro ty, které hledají rychlejší a jednodušší řešení. Další výzvu představuje komplexní škálování a správa clusteru. I když Elasticsearch podporuje clusterování pro zvýšení dostupnosti a výkonu, správa a optimalizace clusteru může být složitá. To zahrnuje vyvažování zátěže, tzv. load balancing, zajištění vysoké dostupnosti, tzv. high availability a řešení problémů s výkonem, které mohou vyžadovat specializované dovednosti. ELK stack je skvělé složení pro monitoring několika různých prostředků, avšak vzhledem k veliké zátěži na systém a náročné konfiguraci včetně následné údržby, je vhodný spíše pro korporátní prostředí. Z tohoto důvodu není v rámci této práce ELK pro následnou implementaci doporučen. [78]

#### 3.6.6.2 Sentry

Sentry je open-source nástroj pro sledování errorů a výkonu aplikací, tzv. performance monitoring v reálném čase. To umožňuje především vývojářům snadno identifikovat, sledovat a opravovat problémy s aplikacemi ve všech etapách vývojového cyklu. Sentry dokáže u již zmíněných errorů získávat spoustu informací včetně stack trace, informací o záznamech akcí uživatelů nebo třeba historii změn, což usnadňuje rychlé zjištění příčin problémů. Integrace Sentry do aplikace je obvykle jednoduchá. [79]

Co však není zcela jednoduché je konfigurace tohoto nástroje. Včetně nasazení v Dockeru se Sentry skládá asi ze 20 containerů, což představuje oproti předešlým uvedeným řešením obrovskou zátěž na hostitelský systém. V tomto případě by bylo potřebné mít vyčleněný jeden server pouze pro nasazení tohoto nástroje. To by však vedlo ke zvýšení nákladů a zapříčinění o to složitější integrace. Sentry je náročný na systém také z důvodu obrovského množství informací a dat, které z aplikací získává. To by mohlo být bráno i jako výhoda, ne však pro malý podnik, ale stejně jako v případě ELK, spíše pro korporátní prostředí. Toto jsou nejzásadnější důvody, proč nástroj Sentry také není pro implementaci v této práci doporučen. [80]

#### 3.6.6.3 Nagios

Dalším analyzovaným nástrojem je Nagios. Jedná se o široce používaný open-source monitorovací systém pro sledování síťových služeb, systémů a jejich komponent. Díky své flexibilitě a možnosti rozšíření pomocí dostupných pluginů umožňuje Nagios sledovat téměř jakékoliv aspekty systémů a jejich komponent, včetně serverů, switchů a aplikací. Nejčastěji tedy nástroj umožňuje pravidelnou kontrolu stavu monitorovaných zdrojů pomocí externích pluginů. Tyto pluginy vracejí stavy zdrojů zpět do systému Nagios. Na základě těchto informací může Nagios generovat také alerting, pokud jsou detekovány problémy nebo pokud se stav zdrojů změní. Funkce Nagiosu zahrnují schopnost plánovat kontroly zdrojů v předem definovaných časových intervalech, opakovaně ověřovat problémy před generováním upozornění, a definovat sítě závislostí mezi monitorovanými zdroji. To umožňuje efektivnější rozhodování o tom, kdy generovat alerting. Ten je možné nastavit prostřednictvím různých kanálů, včetně e-mailu, SMS nebo jiných integrovaných systémů. [81]

Příliš osekaná základní verze bez přidaných pluginů je u tohoto nástroje jednou z hlavních nevýhod. Tato základní verze totiž nenabízí moc možností monitoringu. Avšak v případě rozšíření o několik požadovaných pluginů se podstatně zvětšuje komplexnost systému, která je zbytečně náročná na konfiguraci i údržbu. Kromě toho Nagios také neposkytuje přehledné a efektivní UI zobrazení. To je velikou nevýhodou, jelikož tento nástroj není podporován Grafanou pro možné přenesení získaných dat do efektivnějšího vizualizačního nástroje Grafana. To je však pro tuto práci dosti limitující, neboť v rámci sjednocení je primárně Grafana upřednostňována pro vizualizaci u typů monitoringu, kde je možné jejich vizualizaci přenést do Grafany. Toto jsou tedy primární důvody, kvůli kterým ani nástroj Nagios není doporučen pro následnou implementaci ve vlastní práci. [82]

#### 3.6.6.4 Zabbix

Zabbix je také open-source monitorovací SW založený na podobném modelu jako předešlý Nagios. Avšak Zabbix slouží především k monitoringu stavů různých síťových služeb, serverů a další síťové infrastruktury. Na poměry IT vznikl před dlouhou dobou začátkem tisíciletí a od té doby si vybudoval pozici jednoho z vedoucích nástrojů v oblasti IT monitoringu. Jeho hlavní síla tkví v možnosti adaptace na různorodé prostředí, od malých

podniků až po velké korporace s komplexními sítěmi. Jeho architektura je navržena tak, aby byla schopna zvládnout velké množství dat generovaných monitorovanými zařízeními. Toho dosahuje díky efektivnímu zpracování získaných dat, které minimalizují zátěž databáze nástroje a samotné síťové komunikace. Zabbix je také vybaven flexibilním systémem alertingu, který umožňuje definovat komplexní pravidla pro notifikace a automatické akce založené na různých událostech. [83]

Přestože tento nástroj existuje už dlouhou dobu, za kterou dostál svého jména, i zde je možné najít několik nevyhovujících prvků pro tuto práci. Tím hlavním je zejména to, že v převážné většině firma Sabo spravuje zejména VPS a nikoli fyzické servery. O tento pokročilý síťový monitoring včetně jeho komponent se tedy nemusí tolik starat. Veliká část toho je zajištěna poskytovateli VPS. K většině dat týkajících se síťových prvků by tedy nebylo ani možné se dostat. Avšak u základních síťových prvků, které zde přeci jen monitorovat lze, už tento scénář pokrývá nástroj Prometheus, který byl již doporučen v kapitole výše. Ve srovnání právě s nástrojem Prometheus, nebo jinými nástroji Zabbix, neumožňuje monitorovat větší množství odlišných prvků a také podporuje oproti ostatním pouze dosti omezený počet integrací s třetími stranami. Zároveň je Zabbix poměrně náročný na zdroje, což v konečném důsledku může znamenat i zvýšení nákladů. Z těchto důvodů nástroj Zabbix není také doporučen pro následnou implementaci. [82]

#### 3.6.6.5 Splunk

Posledním nástrojem analyzovaným v této práci je Splunk. Jedná se o výkonný nástroj pro shromažďování, analýzu a vizualizaci dat, zejména logů generovaných sledovanými zdroji. Zaměřuje se na logování dat z různých systémů, jako jsou aplikace, servery nebo síťová zařízení. Jeho hlavním úkolem je umožnit uživatelům rychle najít, sledovat a řešit problémy v IT infrastruktuře a zabezpečit obchodní operace. Splunk toho dosahuje sběrem a indexací velkého množství dat v reálném čase. To poskytuje možnost provádět složité dotazy, analýzy a generování reportů. Díky své flexibilitě a široké škále použití pomáhá firmám zlepšovat výkon, zajistit efektivnější zabezpečení a minimalizovat provozní rizika. [84]

Jednou z hlavních nevýhod je velice obtížná až nereálná optimalizace vyhledávání a sběru dat. To by však zejména vzhledem k množství objemu dat, které Splunk získává, bylo více než potřebné. Je možné říct, že optimalizace se zde stává spíše teoretickou filozofií,

než že by ji bylo reálné doopravdy provést. Dále také kvůli velkému množství získávaných dat může být Splunk velice náročný na systém, a tím také zvětšovat možné náklady na VPS. Další nevýhodou může být také spolehlivost. Ta v konečném výsledku u samotných dashboardů v rámci vizualizace není dobrá. Toto jsou důvody, na základě kterých tento nástroj nelze doporučit pro následnou implementaci ve vlastní práci. [85]

# 4 Vlastní práce

## 4.1 Výběr podniku

Pro tuto práci byla vybrána firma s názvem SABO Mobile IT s.r.o. (dále jen Sabo), IČO: 05680581, zabývající se vývojem webových a mobilních aplikací pro průmyslové využití. Sabo je původem německá firma se zastoupením v Česku. Působí na trhu teprve od roku 2011, kdy ji založili nynější majitel firmy Ing. Josef Sabo a nynější jednatel Ing. Thomas Sykora. V dnešní době má firma tři pobočky, a to v německém Badenu, Praze a Olomouci. Jedná se o podnik malého sektoru s přibližně 40 zaměstnanci.

Sabo je unikátní firmou díky stálému vývoji inovací, zejména tam, kde je IT propojeno s technologií ve strojích, sítích nebo výrobních procesech. Zabývá se zejména HMI, IoT, umělou inteligencí, inteligentními asistenty, plánováním digitálních továren. Firma je většinou schopna vytvářet kompletní řešení sama za sebe, bez pomoci jiných firem, neboť má zastoupení zaměstnanců ze všech potřebných oborů jako UX / UI, front end, back end, testing, DevOps apod.

Firma pracuje na zajímavých projektech jako DPA (digital planning assistent) pro automobilového výrobce Audi, IoT řešení profesionálního pečení SBS (shop baking suite) pro výrobce pečicích jednotek MIWE, mobilní ovládání pro výrobce čerpadel SKF, dálkový monitoring pro výrobce čerpadel KSB, vývoj multifunkční IoT platformy pro firmu pomáhající restauračním zařízením s monitoringem TANIX nebo třeba její vlastní produkt s názvem SABOT. Jedná se o umělou inteligenci v podobě voice bota s využitím právě například u pečicích jednotek MIWE.

### 4.1.1 Analýza současného stavu prostředků v podniku

S tolika projekty firma samozřejmě také spravuje veliké množství IT prostředků, které již není možné efektivně sledovat manuálně. Co se serverů týče, firma jich spravuje přes sto, ať už vlastních nebo klientských. Zároveň Sabo spravuje VPS servery od různých poskytovatelů, ale i fyzické servery klientů. Drtivá většina serverů běží na OS Linux. Ve většině případů Ubuntu 16.04 – 22.04. Avšak jsou zde i výjimky. Pro klienta Comtac spravuje firma dva Linux servery distribuce Debian. Dále pak v rámci několika interních projektů a pro klienta Miwe spravuje firma také několik Windows serverů. Většina těchto

serverů je v podobě virtuálních VPS, zejména od poskytovatelů Contabo, Aruba a AWS v různých konfiguracích co se týče počtu jader CPU, velikosti RAM a velikosti disku, což se odráží na ceně. Tyto servery se tak pohybují měsíčně v různých cenových relacích od 5 $ měsíčně do 50 $ měsíčně. Avšak firma také spravuje, jak již bylo zmíněno, i servery fyzické, a to konkrétně pro klienta Aseko. Pro podrobnější přehled všech aktuálně spravovaných serverů firmou Sabo je vytvořena následující tabulka:

| Klient | Forma serveru | OS - distribuce | Počet |
|---|---|---|---|
| Sabo | VPS | Ubuntu, Windows | 18 |
| Aseko | Fyzické servery | Ubuntu | 24 |
| Kirsch | VPS | Ubuntu | 4 |
| Miwe | VPS | Ubuntu, Windows | 4 |
| Dami | VPS | Ubuntu | 6 |
| Volkswagen | VPS | Ubuntu | 6 |
| Harmonelo | VPS | Ubuntu | 4 |
| Sovto | VPS | Ubuntu | 12 |
| Comtac | VPS | Debian | 2 |
| Gesys | VPS | Ubuntu | 3 |
| CrowningArts | VPS | Ubuntu | 2 |
| Audi | VPS | Ubuntu | 6 |
| Ostatní | VPS | Ubuntu | 16 |

*Tabulka 1 – Přehled serverů spravovaných firmou Sabo*

Dále v rámci těchto serverů firma pracuje s několika stovkami containerů převážně v Dockeru. Zde není úplně snadné říct přesný počet containerů spravovaných firmou Sabo, neboť jejich počet je poměrně variabilní. V Docker containerech jsou nasazeny prakticky všechny aplikace a jiné services, se kterými firma pracuje. Ať už se jedná o produkční BE nebo FE aplikace interní nebo klientské, tak i o nástroje a služby většinou interní pro hladký chod organizace. Ty jsou často převzaté z veřejně dostupných Docker image a jsou jimi například právě monitorovací nástroje. Je skutečně nereálné utvořit konkrétní přehled počtu

containerů na server či klienta vzhledem k variabilitě tohoto počtu. Avšak odhadem se jedná v průměru o 10 až 15 containerů na každý server. To při aktuálním počtu 107 serverů, který je však také relativně variabilní, dělá odhadovaný počet Docker containerů spravovaných organizací 1070 až 1605.

Dále je možné rozdělit tyto containery na buďto jen pomocné k jiným službám, které většinou nejsou publikované ven nebo na právě na takové, které zastupují nástroje, aplikace a jejich FE části a jsou publikovány ven. Tyto FE části jsou běžně publikovány na vlastních DNS. Pro tyto DNS má firma konvenci druhého řádu sabo-gmbh.de s připojením subdomény třetího řádu, jako například tool.sabo-gmbg.de. Ne všechny tooly a aplikace však používají tuto doménu druhého řádu. Především produkční a jiné aplikace využívají domény druhého řádu klienta, jako například aseko.com nebo úplně jiné názvy bez konvenčních domén druhého řádu dle požadavků. Nejen k těmto doménám publikujícím containery ven používá firma nejčastěji Nginx server, případně u některých aplikací dle potřeb Apache. DNS spravovaných firmou je také veliké množství. Odhadem je možné uvést, že cca každý osmý nasazený Docker container je publikovaný ven na vlastní DNS. Při již zmíněném odhadovaném počtu containerů by se tedy jednalo o 133 až 200 publikovaných DNS. Ne všechny však spravuje přímo firma Sabo.

Dále je také dobré zmínit infrastrukturu komunikačních kanálů, které firma Sabo interně používá. Zde se jedná v drtivé většině o služby Microsoft office 365. Z těchto aplikací se používá pro komunikaci zejména Teams. To je důležité zmínit vzhledem k možnosti alertingu monitorovaných dat do komunikačního kanálu. Dále kromě Teams firma používá pro komunikaci spíše externí také Slack.

Firma Sabo také používá své vlastní instance VPN nasazené stejně jako většina ostatních služeb a aplikací v Dockeru. Konkrétně firma používá VPN společností OpenVPN a WireGuard. V rámci WireGuard VPN má firma vytvořených do zásoby 100 peerů, které reprezentují jednotlivé VPN certifikáty. Tato instance je určena především pro distribuci všem zaměstnancům firmy v rámci bezpečnostních norem. Je bez problému použitelná na všechny možné druhy OS mobilních i desktopových pomocí WireGuard clientů viz. https://www.wireguard.com/install/. Dále však firma používá také OpenVPN především pro zaměstnance, kteří potřebují přistupovat k serverům, které mají restricted firewall UFW pouze na adresu této VPN a pro nástroje a services, které potřebují v rámci vývoje také

pracovat s touto VPN. Pro zaměstnance, kteří používají OpenVPN, je možné stejně tak stažení clientů na všechny možné OS tzv. OVPN Connect z https://openvpn.net/client/.

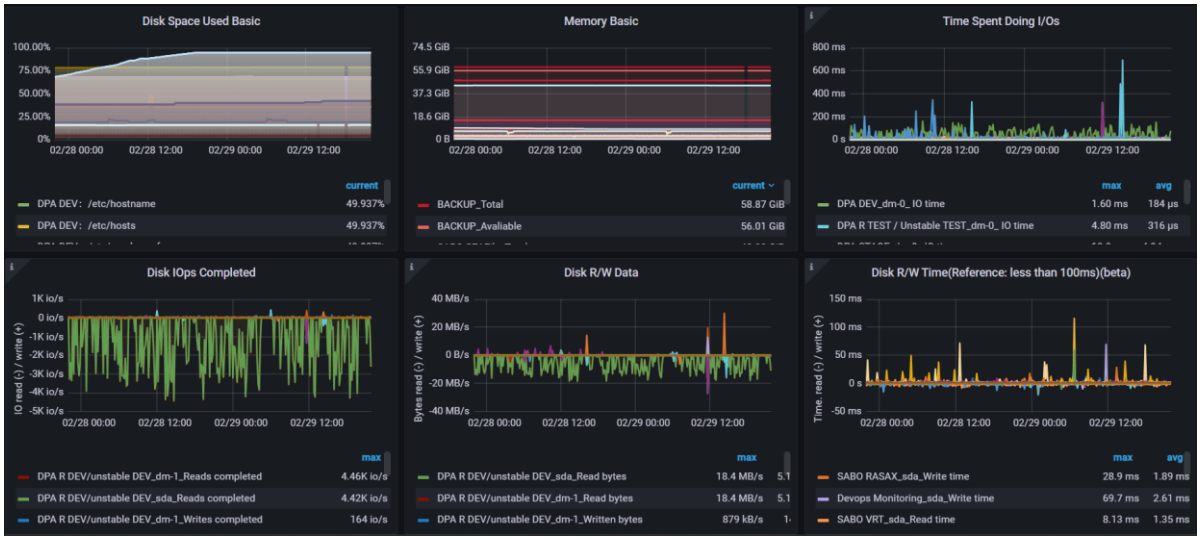
### 4.1.2 Aktuální stav monitoringu v podniku

Jak již bylo zmíněno, Sabo je inovativní novou firmou, tedy již měla jistou formu monitoringu hotovou. Zdaleka však ne tak optimální a automatizovanou, jak by bylo potřeba. Vzhledem k obrovskému množství serverů již firma využívala Grafanu pro vizualizaci jejich stavu. Tato Grafana je publikovaná na doméně druhého řádu, jak již bylo zmíněno dle konvence sabo-gmbh.de. Na žádost firmy však nebudou v této práci zmiňovány konkrétní názvy DNS, hostů, containerů apod. Avšak firma používala stále dnes již zastaralou verzi Grafany, a to konkrétně verzi 8.2.0. Dále k tomuto účelu firma již využívala Prometheus opět na DNS sabo-gmbh.de. Prometheus tahal metriky ze serverů pouze pomocí Docker image pro klasický node exporter a to prom/node-exporter. Navíc ne na všech serverech byl node exporter zaveden. Navíc nebyl zaveden žádný alertovací systém, který by upozornil zaměstnance, především sektoru DevOps na případný problém.

Konkrétně byl tedy zaveden pouze jeden validní monitorovací dashboard Linuxových serverů s defaultním dashboardem převzatým z Grafana Labs dashboards, který nabízí širokou škálu předdefinovaných dashboardů pro Grafanu. Zde se konkrétně jednalo o dashboard s názvem Node Exporter for Prometheus Dashboard English Compatibility Version, který již ani není na Grafana Labs dostupný. Tento dashboard nabízí přehled převážně grafových panelů, pár stats panelů a jeden tabulkový panel informujících o stavu uptime, CPU, RAM, disku, networku a jejich případných variací. Na první pohled se zdá, že by byl tento dashboard pro přehled stavu serverů dostačující, avšak je vytvořený velice nepřehledně. Panely jsou rozházeny v defaultním stavu bez přizpůsobení. Některé panely dokonce nejsou přesně konfigurované na daný node exporter, jelikož vykazují status no data, to znamená, že požadované metriky nebyly nalezeny. Také způsob filtrace není úplně přehledný a neobsahuje ani variantu All pro JOBy, kterým byly dané servery přiřazeny.

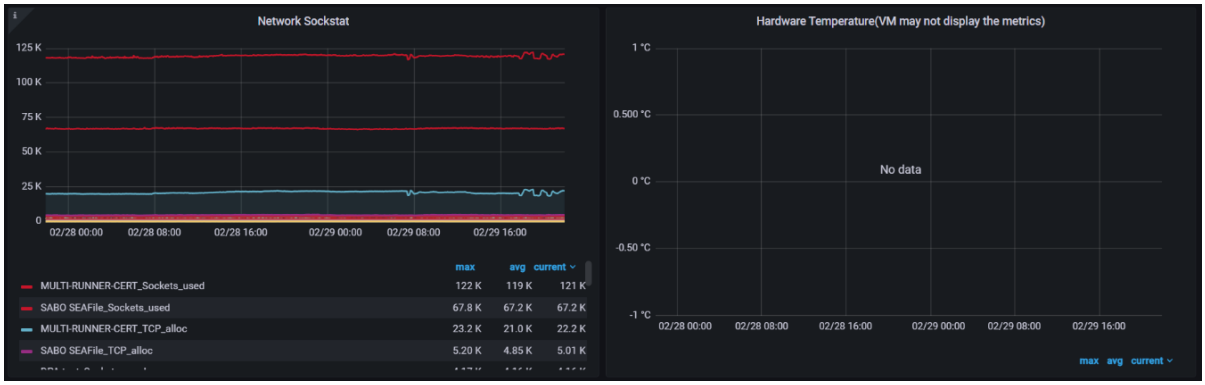

*Obrázek 1 – původní dashboard serverů firmy Sabo 1/3*

Z prvního obrázku původního dashboardu je vidět filtr nahoře, kde je možnost zvolit název JOB, který byl serverům přiřazován v rámci konfigurace *prometheus.yml* podle toho, k jakému klientovi servery patří, případně pokud se jedná o interní servery firmy v JOBu s názvem Sabo_servers. Dále Host, kde je možný výběr serverů, avšak pouze podle jejich ID, což je v praxi nepoužitelné. A následně IP s adresami daných serverů. Pod tím se nachází samotné panely. Nejprve již zmíněné typu stats s přehledem uptime, RAM, CPU, CPU Cores, Used RAM / SWAP / disk. Tyto panely jsou však relevantní pouze po výběru konkrétního serveru. V opačném případě, jak je vidět na *Obrázku 1* se hodnoty všech serverů pro danou skupinu JOB sčítají, což utváří nesměrodatný a relativně nic nevypovídající přehled. Dále se zde nachází tabulka s přehledem diskových jednotek a jejich kapacity / využití. A následně grafy zaměřující se na přehledy disku, CPU a networku. Bohužel i rozložení těchto panelů je tak neefektivní, že se z něj data získávají stěží.




*Obrázek 2 – původní dashboard serverů firmy Sabo 2/3*

*Obrázek 2* poukazuje na pokračování původního dashboardu, kde jsou již pouze další grafové panely týkající se RAM a disku. Opět zde můžeme mluvit o příliš malých a relativně nepřehledných panelech.




*Obrázek 3 – původní dashboard serverů firmy Sabo 3/3*

Poslední *Obrázek 3* původního dashboardu již vyobrazuje pouze zbytek přehledu týkající se networku a teploty HW, který však ani nedostává metriky z node exporteru, čili se jedná o nic nevypovídající panel s hodnotou no data.

## 4.2 Určení use cases a stanovení potřeb

Je tedy zřejmé, že firma Sabo potřebuje efektivnější monitoring spravovaných prostředků. Bylo navrženo efektivnější zpracování dashboardu s přehledem serverů. Ten by měl být zpracován taktéž z předdefinovaného dashboardu Grafana Labs, avšak takový, který bude lépe sedět požadavkům firmy Sabo, aby byl přehlednější, na první pohled by měly být zřejmé nejzásadnější údaje a customizovat dashboard dle metrik z nástroje Prometheus, kterému vytahuje a odesílá metriky opět stejný image prom/node-exporter tak aby mohly být zachovány oba panely a zároveň aby se na serverech nehromadily duplicitní containery node exporteru.

Dále pak firma požaduje dashboard obsahující alerting do interního kanálu Teams, který by se týkal varování před nebezpečně vysokým využitím disku. Neboť zrovna disk usage je u velikého počtu serverů firmy Sabo problém. Mnoho serverů se nepřetržitě plní daty a pokud dojdou k 100% využití disku, veškeré containery na takovém serveru přestávají fungovat a pro odstranění tohoto problému je nejprve nutný debugging v rámci hledání příčiny kompletního zaplnění kapacity serveru. Často se totiž jedná o zastaralé soubory s daty, které již nejsou potřeba a problém je tak možné vyřešit pouhým vymazáním těchto souborů. Zde by monitoring zajistil to, že zaměstnanci IT oddělení DevOps obdrží notifikaci týkající se blížícího zaplnění disku serveru tak, aby měli dostatečný čas si problému všimnout a vyřešit ho, než se server zcela zaplní a jeho služby a aplikace tak přestanou fungovat.

Mimo serverů je zde další požadavek firmy na sledování a alerting expirace SSL certifikátů. Jak již bylo zmíněno mezi prostředky firmy, Sabo vlastní a spravuje opravdu veliké množství DNS z nichž většina používá SSL certifikáty k zabezpečení šifrování dat na protokol https. Tyto certifikáty mají však svoji expiraci, většina je nastavena na rok a poté je třeba SSL certifikáty obnovit. Z tohoto důvodu je požadováno rozšíření monitoringu o monitorování expirace SSL certifikátů s předběžnými aletry opět do komunikačního kanálu Teams pro IT zaměstnance oddělení DevOps, pro možnost vytvoření nových certifikátů, než dojde k expiraci. To by mělo opět zamezit výpadkům služeb a aplikací, které na daných DNS fungují.

Další požadavek je na vytvoření logování záznamů aplikací (převážně BE). To potřebují především vývojáři BE aplikací pro možné efektivní debuggování a rychlejší vývoj

těchto aplikací. Případně může být tento monitorovací systém použit i na FE aplikace a databáze, zde to však ve většině případů nedává takový smysl, navíc konkrétně u databází se jedná o obrovské množství záznamů jen těžko filtrovatelné a přístupné pro případné nalezení problémových úseků. Tento monitorovací systém by měl být použit pouze pro určité konkrétní aplikace, dle požadavků firmy Sabo nebo jejich klientů.

Další požadavek na součást monitoringu a efektivní správy Docker containerů vychází taktéž od vývojářů BE i FE. Ty by chtěly vidět Docker logy jejich aplikací a v některých případech pro efektivnější práci i možnost úprav v těchto Docker containerech. Avšak vývojaři nemohou mít přístup na servery jako takové, což vede k vytvoření alternativního řešení vzdáleného přístupu k Docker containerům daných serverů, avšak pouze k containerům, ne ke konfiguraci serverů jako takových.

Další požadavek je velice specifický, a to od klienta Comtac. Mají určité aplikace vystavené na třech různých portech, avšak toto prostředí není úplně stabilní. Z tohoto důvodu by chtěli monitorovací systém těchto určitých portů na zjištění jejich uptimu s případným alertingem v případě výpadků. Pro nejefektivnější možné řešení se zde nabízí dotažení automatizace až do konce k vyřešení problému. Tedy po zjištění, že je určitý port tzv. down neboli nefunkční, klient a pracovníci DevOps dostanou notifikaci o tomto stavu. Následně se automatizovaně spustí script pro restartování této aplikace, což by mělo ve většině případech problém vyřešit a pracovníkům DevOps včetně klienta přijde druhá notifikace o následné opravě, pokud proběhne úspěšně.

Další požadavky na monitoring jsou od klienta Kirsch. Ten požaduje hned několik věcí. Za prvé je zde potřeba efektivní monitorování prostředků 4 serverů, na kterých jsou nasazeny jednotlivá prostředí Kirsche. Těmito prostředími jsou DEV, TEST, STAGE, PROD. Požadavek je tedy na monitorování využití prostředků těchto serverů jako klasické CPU, RAM, Disk, network atp. Dále je zde požadavek i na monitoring jednotlivých Docker containerů, což by mohlo být použitelné na všechny servery firmy Sabo, avšak při tak obrovském počtu Docker containerů by se jednalo o velice nepřehledné a zbytečně resource beroucí monitorování. Z tohoto důvodu je zde požadavek monitorování Docker containerů pouze u klienta Kirsch. Mimo klasický monitoring využití prostředků Docker containerů je také požadavek sledování uptime včetně alertingu opět do kanálu Teams pro IT zaměstnance oddělení DevOps a přiřazené vývojáře klientu Kirsch. Poslední požadavek klienta Kirsch je

zde na vypracování měsíčních reportů stavu jejich aplikací, a to nejen v rámci IT prostředků, ale také například co se týká kvality zabezpečení Nginx serveru.

Posledním požadavkem na monitoring, který bude zpracován v rámci této práce je monitoring VPN certifikátů. Zde ovšem není nutné monitorovat VPN, které používají přímo zaměstnanci, neboť ti sami vidí v nainstalovaných clientech, jestli VPN funguje nebo ne. Avšak v případě instance OpenVPN, jejíž VPN certifikáty jsou nasazovány i některým vyvíjeným aplikacím a servicám je monitoring potřebný. U takových certifikátů totiž není možné sledovat, zdali opravdu fungují. V případě takové nefunkční aplikace totiž může být příčina problému jak v kódu aplikace, tak právě i ve VPN. Z tohoto důvodu bude nasazen OVPN monitoring, který bude monitorovat všechny aktuálně spuštěné VPN certifikáty a jejich vlastnosti.

## 4.3 Návrh architektur monitorovacího řešení

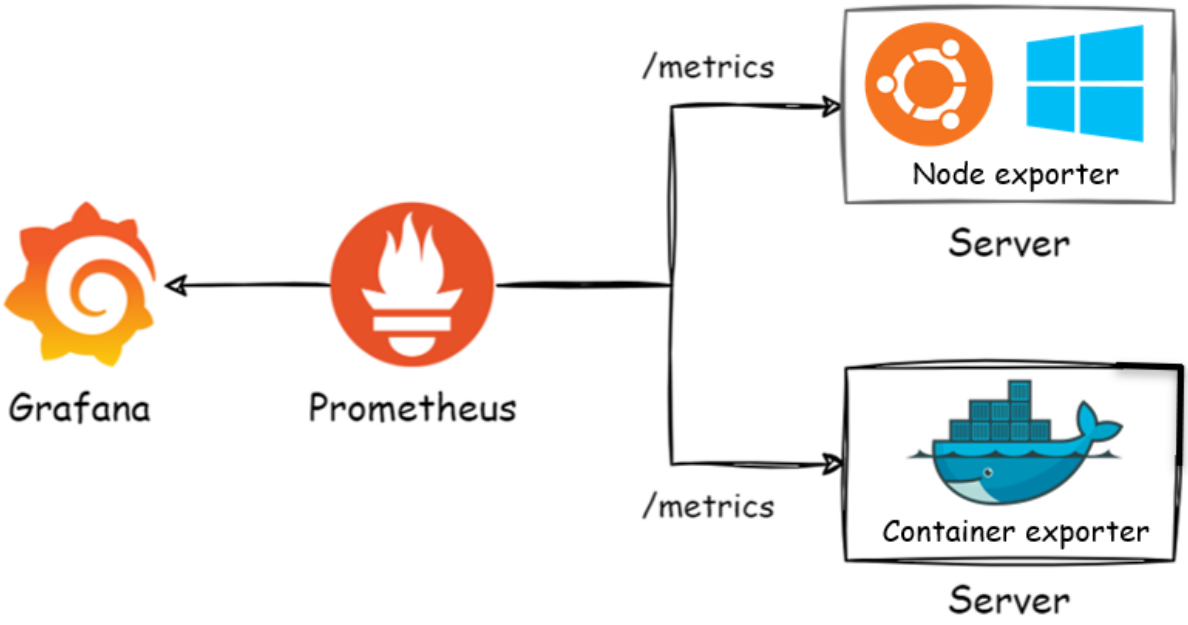
### 4.3.1 Vizualizace – Grafana

Jak bylo výše zmíněno, firma aktuálně používá Grafanu na DNS druhého řádu sabo-gmbh.de. Zde se tedy již nachází jeden produkční dashboard a pár dalších zkušebních. Pro tuto práci a pro firmu byly vytvořeny DNS druhého řádu devops-monitoring.com což je s danými nástroji použito s doménou třetího řádu jako tool.sabo-gmbh.de pro každý z nástrojů zvlášť. Avšak nová monitorovací řešení nebudou vytvářena pouze na DNS devops-monitoring.com, ale také na původní sabo-gmbh.de dle požadavků firmy, aby byla zátěž rovnoměrně rozložena do dvou prostředí a zároveň aby se nemuselo vytvářet více instancí pro alerting. Co se tedy Grafany jako takové týče, zde půjde především o přidání dalších dashboardů dle požadavků, případně upravení předešlých pro větší efektivitu. Dashboardy by měly být přehledné případně obsahovat přehlednou obecnou část na prvním místě a následně zbytek panelů pod tím především pro zaměstnance DevOps, kteří často potřebují více podrobné informace o monitorovaném prostředku. Dále budou přidány aletry především do firemního komunikačního kanálu Teams, případně do Slacku. Zároveň bude nasazena Grafana novější verze na DNS devops-monitoring.com.

### 4.3.2 Architektura s nástrojem Prometheus a exportery

Firma aktuálně tedy využívá Prometheus na DNS druhého řádu sabo-gmbh.de a node exportery pro export metrik ze serveru do nástroje Prometheus. Tato metoda je vcelku

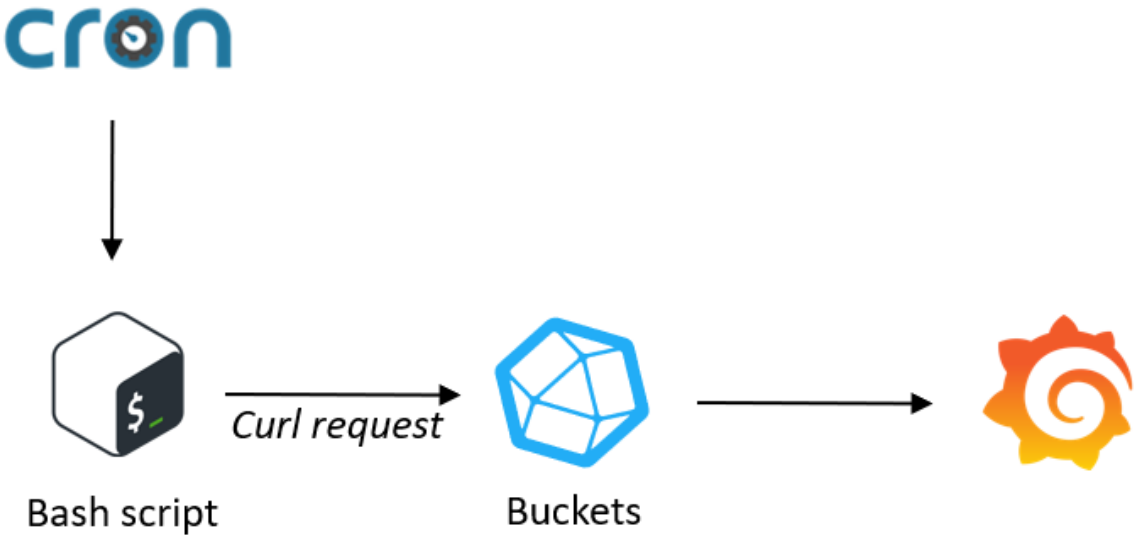
efektivní tak bude také znovu použita, avšak vylepšena. Pro tuto práci a pro bude opět přidán Prometheus na DNS druhého řádu devops-monitoring.com což bude opět použito s doménou třetího řádu jako tool.sabo-gmbh.de. Avšak nová monitorovací řešení nebudou vytvářena pouze na DNS devops-monitoring.com, ale také na původní sabo-gmbh.de dle požadavků firmy, aby byla zátěž rovnoměrně rozložena do dvou prostředí a zároveň aby se nemuselo vytvářet více instancí pro alerting. Vzhledem k již nasazeným containerům s imagem prom/docker-container na několika serverech budou tedy stejně tak použity tyto containery pro získávání metrik z OS Linux. Tento image je pro dané potřeby naprosto dostačující, akorát bude přidán a nasazen na servery, kde ještě chybí. Podle toho bude následně upravena konfigurace *prometheus.yml*. Následně pak pro specifické požadavky především klienta Kirsch bude nasazen na některé servery také container exporter pro možné získávání metrik s daty týkajících se jednotlivých nasazených containerů na serveru. Na základě dobrých zkušeností s imagem prom/node-exporter se zde bude konkrétně jednat o image prom/container-exporter.




*Obrázek 4 – schéma návrhu řešení s nástrojem Prometheus*

### 4.3.3 Architektura s nástrojem InfluxDB a bash scripty

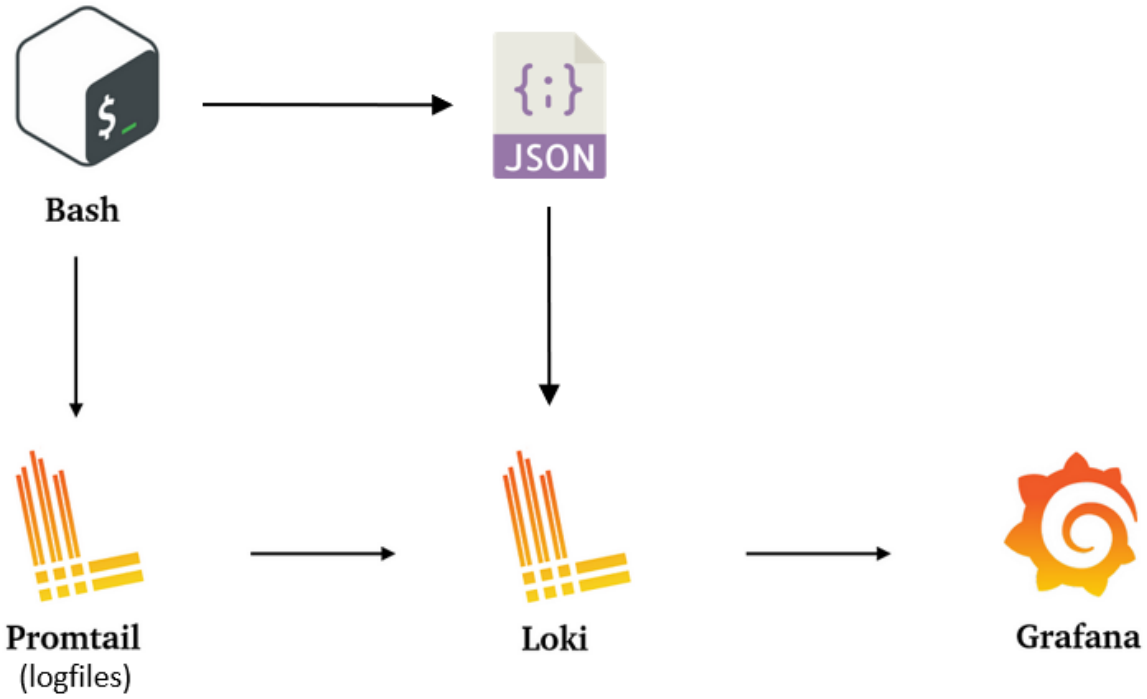
Dalším navrhovaným řešením je použití nástroje InfluxDB. Zde se jedná o velice specifické a customizované řešení pro specifické požadavky především klienta Comtac a Kirsch. Mimo to by se touto metodou měl vytvořit i interní monitoring SSL certifikátů. InfluxDB je tedy specifická time series databáze, která bude nahrazovat Prometheus z předchozího řešení. Zde se totiž nebude jednat o získávání klasických dat z imagů exporterů, avšak vzhledem k specifičnosti požadavků zde bude potřebné vytvoření vlastních bash scriptů pro získávání požadovaných dat. Konkrétně se tedy bude jednat o scripty získávající data uptimu portů aplikací klienta Comtac s případným resetováním portu, u kterého bude zjištěno že je down. Dále případné dodělání monitoringu Docker containerů klienta Kirsch, zde totiž metoda s použitím container-exporteru je nedostačující na požadovaný alerting uptimu containerů. Mimo to by měl být touto metodou zpracován také monitoring všech DNS a jejich SSL certifikátů s alertingem několik dnů před expirací. Tyto scripty bude následně spouštět služba cron dle předdefinovaného pravidelného časového opakování. To následně povede díky curl requestům v scriptech k automatizovanému zasílání dat do time series databáze InfluxDB, kde se budou data skladovat v před definovaných bucketech. Tyto buckety oddělují time series databáze od sebe. InfluxDB následně data předá opět Grafaně, kde bude možné pomocí bucketů tyto data filtrovat.




*Obrázek 5 – schéma návrhu řešení s nástrojem InfluxDB*

### 4.3.4 Architektura s nástrojem Loki

Dalším řešením, které bude zavedeno především v rámci původní Grafany na DNS sabo-gmbh.de je získávání a zobrazování logů především BE aplikací nasazených v Dockeru. Pro tento účet je využíván nástroj Loki, který je možné chápat jako specifické úložiště právě těchto logů. Loki nebude však oproti ostatním nástrojům nasazen na DNS, neboť se zde nepracuje s žádným UI a nemá to tedy smysl. Bude nasazen na serveru určeném přímo pro nástroj Loki vzhledem k velikému objemu dat v podobě logů aplikací. Tím bude zamezeno pádu jiných aplikací v případě dosažení maximální kapacity disku. Ze serverů, kde se nachází logované aplikace však bude Loki získávat data pomocí dvou odlišných způsobů. Prvním je nástroj Promtail přímo kompatibilní s nástrojem Loki, neboť je od stejného distributora. V tomto případě Promtail vytváří soubory tzv. logfiles, které jsou následně podle konfigurace nástroje Loki v souboru local-config.yml distribuovány do Lokiho, který data uchovává a předává je Grafaně. Vcelku je toto řešení o něco šetrnější k zatížení disku, avšak díky ukládání dat v souborech neumožňuje vývojářům efektivní debuggování těchto logů, neboť zde není možné sledovat celé multilines logy v rámci jednoho zápisu. Z tohoto důvodu bude přistoupeno souběžně také k řešení za použití datového formátu vycházejícího z jazyka JavaScript JSON scriptů, které budou zasílat logy aplikace přímo do nástroje Loki bez nutnosti použití logfilů. Tím umožní sledovat celé logy včetně multilines, avšak způsobí náročnější, již tak veliké zatížení disku na serveru, kde bude nasazen Loki. Z tohoto důvodu bude toto řešení použito jen pro určité prostředí aplikací. JSON scripty však budou přímou součástí aplikace, kterou vytvářejí vývojaři. Vzhledem k tomu tyto JSON scripty nebudou vstupem pro tuto práci. Pouze jejich následné zpracování v nástroji Loki a Grafaně.




*Obrázek 6 – schéma návrhu řešení s nástrojem Loki*

### 4.3.5 Monitoring VPN

Instance OpenVPN spravovaná firmou SABO tedy potřebuje monitoring aktuálně aktivních VPN certifikátů. Samotná instance OpenVPN je nasazena v Dockeru skrze image *kylemanna/docker-openvpn*. K tomuto řešení se nabízí s ním spojený monitoring VPN certifikátu pod Docker image *ruimarinho/openvpn-monitor*. Toto řešení je sice od jiného distributora, avšak je přímo uzpůsobeno monitoringu OpenVPN. S využitím napojení přes UDP by tak měl být zajištěn efektivní monitoring aktivních VPN certifikátů.

### 4.3.6 Portainer

Poslední součástí návrhu monitorovacího řešení je použití a nakonfigurování nástroje Portainer. V tomto nástroji bude možné nejen sledování logů Docker containerů bez nutnosti přístupu na server, kde se dané containery nachází, ale také jejich přímá správa. Zde se tedy nejedná pouze o monitorovací nástroj, avšak je možné ho přiřadit k monitorovací struktuře. Bude to zároveň tedy jediné řešení, kde nebude k vizualizaci použit nástroj Grafana. Po technické stránce se zde bude jednat o nasazení Portaineru jako Docker container na serveru

k tomu určeném a následné napojení jeho agentů na servery které budou spravovány. Tito agenti budou také nasazeni formou Dockerových containerů.

## 4.4 Implementace daného řešení

V této kapitole je detailně popsána implementace celého monitorovacího řešení. Avšak vzhledem k doporučenému rozsahu je nutnost většinu konfigurací a obrázků uvést v přílohách. Z tohoto důvodu jsou zde vyobrazeny pouze např. první konfigurace daného jazyka. Co se však obrázků samotných výsledků práce, tedy vizualizací monitorovacích řešení týče, ty jsou všechny zobrazeny zde, přímo v práci.

### 4.4.1 Nasazení Grafany

Jak již bylo tedy zmíněno Grafany jsou nasazeny dvě. Obě jsou na jiných serverech, s tím že je jedna původní, kterou firma již používala a druhá nová v rámci této práce na serveru, kde se nachází prakticky všechny nástroje pod DNS devops-monitoring.com. Původní konfigurace, tím že již existovala zároveň nebude součástí této práce. Avšak ve výsledku se odlišují především jen verzí image Grafany. Obě instance jsou tedy nasazeny jako Docker containery. Zde je konfigurace Grafany devops monitoring v *docker-compose.yml*:

```
grafana:
  image: grafana/grafana
  restart: always
  ports:
    - 127.0.0.1:<PORT>:3000
  environment:
    GF_SERVER_ROOT_URL: https://grafana.devops-monitoring.com
  volumes:
    - ./grafana/grafana:/etc/grafana:rw
    - ./grafana/conf:/usr/share/grafana/conf:rw
    - ./grafana/varlibgrafana:/var/lib/grafana:rw
```

Zde vidíme na prvním řádku název servicy Grafana, dále image Grafana/Grafana bez doplnění verze čili latest. Což v době nasazení byla Grafana v9.5.2. Dále restart always, který je připsán téměř u všech containerů. Kdykoli totiž může kvůli nějaké krátkodobé chybě jako například zaseklý proces, jakýkoli aplikační error nebo zkrátka jen neočekávaný leak

container tzv. shodit do stavu exited. Restart always se tedy staré o to, že po pádu containeru okamžitě daný Docker container sám sebe restartuje. To ve většině případů opět container nasadí do stavu up a aplikace tak spadne na tak krátko chvíli, e to uživatel běžně ani nepozná. Pouze pokud zůstane container ve stavu restarting, jedná se o závažnější problém konfigurace, který se pouhým restartem neopraví. Container tak sám sebe nepřetržitě jen restartuje bez dosažení výsledku. Dále následují porty, vnější port, který je publikován ven je na žádost firmy cenzurován. Vnitřní port je pak obecně daný pro Grafanu jako 3000 a to celé je uzamčeno na IP známé jako localhost 127.0.0.1. V rámci environment proměnných je zde využita pouze proměnná GF_SERVER_ROOT_URL s hodnotou https://grafana.devops-monitoring.com což značí DNS kam bude následně Grafana vystavena skrze Nginx server, neboť s uzamčenými porty na localhost je jinak UI nedostupné. Na závěr jsou nakonfigurované volumes. Zde se jedná o transfer souborů mezi serverem a obsahem Docker containeru. Zde musel být při prvním nasazení manuálně nejprve vytažen celý obsah Grafany a následně jej nastavit přes volumes při opakovaném nasazení. Zde nejde nastavit volumes celého repositáře rovnou neboť volumes mají význam přenesení souborů ze serveru do containeru. Pokud by se na serveru tedy původně nic nenacházelo, vymazalo by to celý obsah containeru. Důvodem, proč je toto u Grafany důležité udělat je například upgrade. Při upgradu dochází ke stavu Docker containeru down. Pokud by tedy nebyly nastavené volumes, došlo by ke ztrátě veškerých dat. Následně je skrze volumes ještě zvlášť vystaven konfigurační soubor Grafany grafana.ini a databáze Grafany na cestě /var/lib/grafana.

O tom, že je Grafana up je možné se přesvědčit ve výpisu containerů docker ps:


*Obrázek 7 – výpis containerů „docker ps -a“*

Zde můžeme vidět Grafanu na 8. řádku odshora ve výpisu containerů. Zároveň se jedná o výpis containerů celého serveru určenému pro devops monitoring. Tedy zde můžeme

vidět také spoustu dalších nástrojů, u kterých bude jejich nasazení blíže rozepsáno níže. Co se sloupců ve výpisu „*docker ps -a*“ týče, je možno vidět na prvním místě CONTAINER ID, na druhém IMAGE, na třetím COMMAND který vyobrazuje použitý spouštěcí command v tomto případě zakomponovaný v image grafana/grafana, na čtvrtém místě CREATED značící, před jakou dobou byl container nasazen, na pátém místě STATUS, kde si právě můžeme ověřit jestli je Grafana up, dále PORTS, kde v tomto případě opět zůstane vnější port cenzurován a poslední sloupec je NAME jakožto název containeru.

Po ověření, že je container Grafany up je možno Grafanu nasadit na DNS. Pro tento účel byl vytvořen a spuštěn konfigurační soubor Nginx serveru viz:

```
server {

        server_name grafana.devops-monitoring.com;

        location / {

        proxy_set_header Host   $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_pass          http://127.0.0.1:<PORT>;
        proxy_read_timeout  90;
        }

    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/grafana.devops-
monitoring.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/grafana.devops-
monitoring.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot

}
server {
    if ($host = grafana.devops-monitoring.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot

        server_name grafana.devops-monitoring.com;

        listen 80;
```

```
        listen [::]:80;
    return 404; # managed by Certbot
}
```

Zde se jedná o klasický Nginx konfigurační soubor obsahující název DNS, pouze základná location / s několika hlavičkami CORS pro správné fungování stránky, proxy pass odkazující na Docker container Grafany, avšak zde je opět cenzurovaný vnější port. Dále je zde otevřen port 80 pro naslouchání na všech IPv4 i IPv6 adresách. A také několik automaticky vygenerovaných řádků consolovým nástrojem Certbot, který vytváří SSL certifikáty. Zde Certbot také vytvořil všechny potřebné soubory a certifikát na cestě /etc/letsencrypt/live/grafana.devops-monitoring.com/.

### 4.4.2 Klasický monitoring prostředků serverů a containerů

#### 4.4.2.1 Nasazení nástroje Prometheus

I Prometheus je tedy opět nasazen jako Docker container. Zde jsou však nasazeny dvě nové instance nástroje prometheus na tomto serveru devops monitoring, jeden klasický a druhý pro klienta Kirsch, níže bude vysvětleno, proč tomu tak je. Po následném nasazení bude pouze hlavní instance publikována na DNS prometheus.devops-monitoring.com. Avšak zde už je tedy také jedna instance nástroje Prometheus vytvořena na DNS prometheus.sabo-gmbh.de. Nasazení již existující instance nástroje Prometheus však také nebude obsahem této práce. Jelikož se nástroj Prometheus nasazuje také na serveru určeném pro devops-monitoring, bude přidán do stejného souboru *docker-compose.yml* jako Grafana. Avšak v této kapitole bude rozebrána pouze část nástroje Prometheus. Kompletní sjednocení souboru *docker-compose.yml* bude zobrazeno v kapitole níže. Zde tedy konfigurace *docker-compose.yml* vypadá následovně:

```
  prometheus:
    image: prom/prometheus:latest
    restart: always
    ports:
      - 127.0.0.1:<PORT>:9090
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
```

```
    - '--web.console.templates=/etc/prometheus/consoles'
    - '--web.enable-lifecycle'
  volumes:
    - ./prometheus:/etc/prometheus:ro
```

Oběvují se zde podobné, ba i stejné prvky jako u Grafany, ty tedy již nebude nutné blíže vysvětlovat. Avšak zde se tedy service jmenuje logicky prometheus. následně je zde použitý image *prom/prometheus:latest* s konkrétním dodatkem verze latest. Zde to byla v době nasazení verze 2.44.0. Opět je zde použit restart always a porty zamčené na localhost. Vnější opět cenzurován a vnitřní tak jak bývá pro Prometheus zvykem 9090. Dále se zde nachází několik spouštěcích commandů, které do upravují konfiguraci nástroje Prometheus při jeho nasazení. Posledním prvkem této konfigurace jsou volumes. Zde hrají zásadní roli neboť na cestě serveru *./prometheus* se nachází soubor *prometheus.yml* který obsahuje hlavní konfiguraci prostředí nástrojů. V této práci však nebude ukázán celý. Zaprvé zde bude mnoho hodnot cenzurovaných, neboť obsahuje adresy všech serverů, vnější porty jejich exporterů, interní názvy hostů, ale především je velice dlouhý a repetitivní, neboť obsahuje všechny servery, které jsou monitorované. Ukázka použitého konfiguračního souboru *prometheus.yml* již tedy vzhledem k rozsahu musí být zobrazena v přílohách viz *příloha A.* Jedná se tedy pouze o krátkou ukázku konfiguračního souboru *prometheus.yml*. Přesto tedy začátek je nezměněný, tam se uvádí v tomto případě nejdřív pár obecných časových konfigurací týkající se metody práce s metrikami. Následně je nadefinovaný alerting s odkazem na container tohoto alertingu, který již musí být cenzurován, neboť je nasazen na jiném serveru. Avšak konfigurace konkrétních alertů se nachází na cestě uvnitř Docker containeru */etc/prometheus/alert.yml*. Dále již následují scrape configy, respektive konfigurace jednotlivých sledovaných serverů přidělených určitým skupinám job name. Tyto job names následně umožní efektivní rozdělování skupin serverů například dle přiřazených klientů pro následně možnou efektivní filtraci ve vizualizaci Grafana dashboardů. Zde je podrobněji ukázán pouze job name s názvem node, který odkazuje sám na sebe. U tohoto job namu je tedy ukázán i host, neboť zde není uvedená IPv4 adresa, ale pouze název containeru node a container exporterů což je dostačující, neboť jsou všechny tři containery nasazené ve stejném networku a odlišují se pouze lokální adresou containerů. Dále je zde viditelný i název instance, jelikož je to právě Devops Monitoring, který vznikl na základě této práce. Dále jsou jen jako příklad ukázány další 3 produkční job namy, které

ale obsahují už jen produkční IPv4 adresy serverů, porty exporterů, kterých se zde objevuje zhruba 4 různé druhy dle toho, o jaký exporter se jedná a názvy instancí neboli produkčních hostů. To jsou všechno informace, které musí být v této práci cenzurovány. Dále je zde vhodné zmínit už jen *metrics_path: /metrics*. To udává location, na které se dané metriky odesílají, což je důležité pro následný monitoring.

Druhá instance nástroje Prometheus je tedy jak již bylo zmíněno pouze pro specifický požadavek klienta Kirsch. Ten, jak již bylo zmíněno, požaduje z Grafana dashboardu vytvářet měsíční reporty ohledně stavu a chodu jejich aplikace na produkčním prostředí PROD. U původní instance je doba, za jakou se budou metriky samy mazat, tzv. time retention, nastavena na 14 dní, což by pro měsíční reporty nestačilo. Avšak je nežádoucí zvětšovat time retention na celý Prometheus, neboť by to znamenalo obrovskou zátěž využití disku pro celý server. Z tohoto důvodu je tedy vytvořena druhá instance nástroje, která má s rezervou time retention nastavený na 60 dní. V *docker-compose.yml* vypadá tedy následovně:

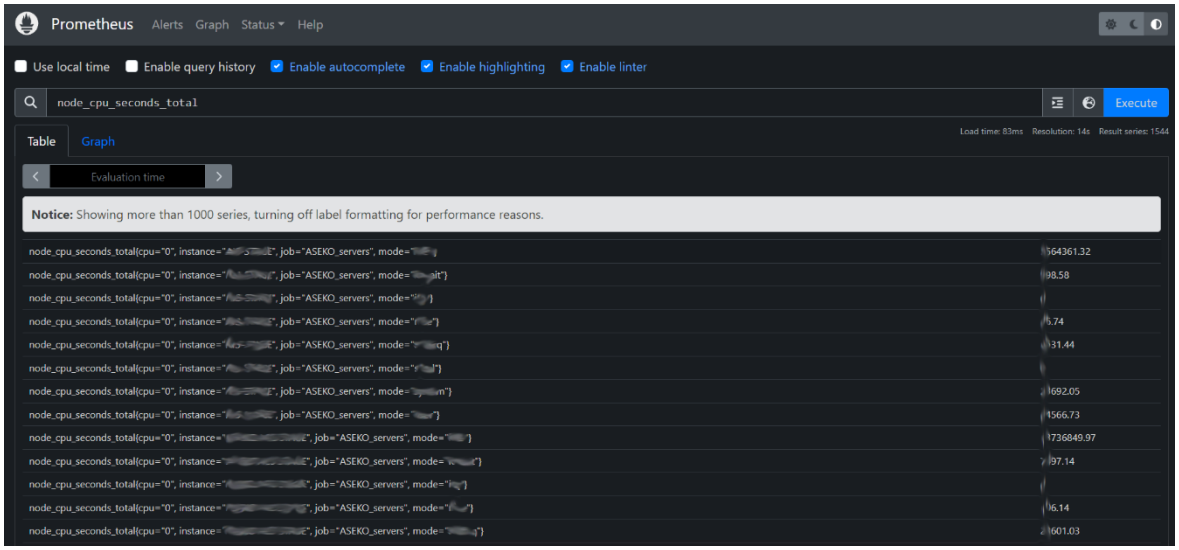
```
prometheus-kirsch-2m:
    image: prom/prometheus:latest
    restart: always
    ports:
      - 127.0.0.1:<PORT>:9090
    command:
      - '--config.file=/etc/prometheus/prometheus.yml'
      - '--storage.tsdb.path=/prometheus'
      - '--web.console.libraries=/etc/prometheus/console_libraries'
      - '--web.console.templates=/etc/prometheus/consoles'
      - '--storage.tsdb.retention.time=60d'
      - '--web.enable-lifecycle'
    volumes:
      - ./prometheus-kirsch:/etc/prometheus:ro
```

Zde jsou vlastně jedinými rozdíly vnější port, který je však cenzurovaný a pak právě 5. řádek commandu - *'--storage.tsdb.retention.time=60d'*, který mění defaultní hodnotu time retention 14d na 60d. Konfigurační soubor *prometheus.yml* je na stejném principu. Pouze jsou zde jiné job namy a jiné adresy a názvy serverů, které by však stejně byly cenzurované.

Nginx konfigurace je zde v mnoha aspektech stejná jako u Grafany, avšak jedna zásadní změna tu je. Nástroj Prometheus jako takový nenabízí žádnou variantu autentizace,

z tohoto důvodu je nutné zde v Nginxu nadefinovat autentizaci přímo na úrovni serveru tzv. basic auth. Rozdíl v konfiguraci tedy vypadá viz. *příloha B.* V této příloze je tedy možno vidět opět nadefinování názvu DNS, následně opět pouze location / s CORS určenými pro Prometheus a proxy pass odkazující na port Docker containeru. Ale objevují se zde navíc řádky basic auth, které definují název autentizace a odkaz na soubor s šifrovanými přístupovými údaji. Nginx konfigurace Prometheus instance určené pro Kirsch je víceméně totožná, pouze s odlišnou DNS, portem containeru a odkazem na soubor basic authu.

Nyní, když je vše nasazené a z *Obrázku 7* v předchozí kapitole je zřejmé, že obě instance jsou nasazeny, je možné zkontrolovat UI nástroje Prometheus:




*Obrázek 8 – ukázka UI vyhledávače metrik nástroje Prometheus*

Na *Obrázku 8* je možné vidět UI přívětivý vyhledávač metrik. Jako příklad byla zadána metrika *node_cpu_second_total* týkající se využití CPU. Po vyhledání této metriky je zobrazeno spoustu řádků, podle poznámky nahoře více než tisíc v tomto případě, které se týkají této metriky. Jsou zde vyobrazeny zvlášť řádky pro všechny možné kombinace instancí, job namů, případně další možnosti filtrace dle konkrétní metriky a jejich hodnoty. Avšak toto UI se zobrazuje po zadání DNS prometheus.devops-monitoring.com bez předané location. Jak bylo výše zmíněno, definovaná location */metrics* pak vyobrazuje kompletní seznam všech metrik, které jsou předávány Grafaně viz. *příloha C.* Následně je také dobré znát možnost zobrazení Targets v nástroji Prometheus. Toto UI zobrazuje všechny skupiny

dle job namů, jim přiřazené servery a statusy těchto serverů viz. *příloha D.* Ne všechny servery jsou vždy UP vzhledem k jejich aktualizaci nebo případným problémům. Toto zobrazení však opět musí být vzhledem k zásadám cenzurované.

#### 4.4.2.2 Nasazení exporterů

Nutnou součástí této metody je tedy nasazení exporterů. Ty tedy zprostředkovávají transport dat ze serverů do nástroje Prometheus. Jedná se čistě o consolové nástroje, bez žádného UI. Jednoznačně nejvyužívanějším je node exporter, získávající metriky z každého serveru. Vzhledem k předchozímu použití firmou Sabo image *prom/node-exporter* bude tedy i na ostatních serverech nasazen právě tento exporter. Jeho konfigurace v *docker-compose.yml* vypadá následovně:

```
node-exporter:
  image: prom/node-exporter
  ports:
    - <PORT>:9100
  restart: always
```

Zde se tedy jedná o naprosto jednoduché nasazení Docker containeru. Tyto containery nejsou uzamčené na IP localhostu, neboť nejsou publikovány ven skrze Nginx a je k nim potřebný přístup z jiného serveru, konkrétně Devops Monitoring. Ostatní prvky, které node exporter obsahuje byly již výše zmíněny tudíž není potřebný detailnější rozbor.

Avšak je zde možnost využití dalších exporterů, druhým nejvýznamnějším, který také stojí za zmínku je container exporter získávající data z containerů nasazených na serveru. Tento container je tedy použit pouze na určitých serverech, kde bylo jeho použití vyžádáno a má následující konfiguraci:

```
container_exporter:
  image: prom/container-exporter
  ports:
    - <PORT>:9104
  restart: always
  volumes:
    - /sys/fs/cgroup:/cgroup
    - /var/run/docker.sock:/var/run/docker.sock
  deploy:
```

```
resources:
  limits:
    memory: 1G
```

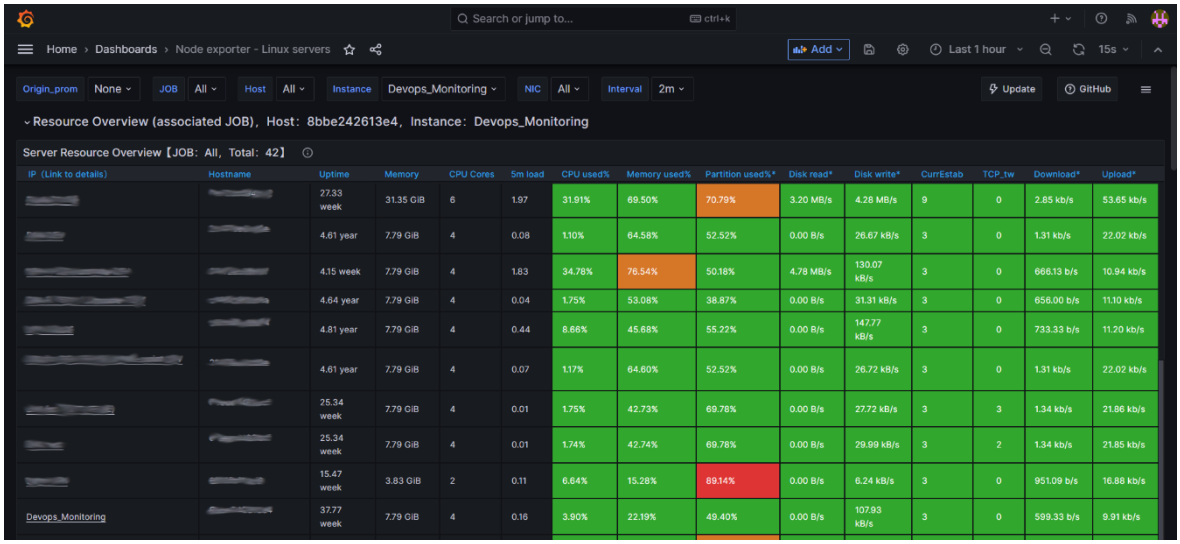
Co zde zejména stojí za zmínku je sekce deploy s limitem RAM nastaveným na 1 GB. Zde byl totiž veliký problém s ne úplně efektivním cachováním dat v rámci tohoto containeru, který způsoboval veliký a nepřetržitý nárůst RAM na serveru. Tak byl přidán tento limit, který způsobuje restart tohoto containeru, jakmile dojde na limit RAM 1 GB. Tím se zamezí dosažení maximálního limitu RAM serveru a pádu ostatních prostředků v tomto důsledku. Volumes které jsou zde dosti specifické jsou defaultní od distributorů image *prom/container-exporter* a zajišťují pouze sběr dat pro metriky.

#### 4.4.2.3 Vizualizace v Grafaně

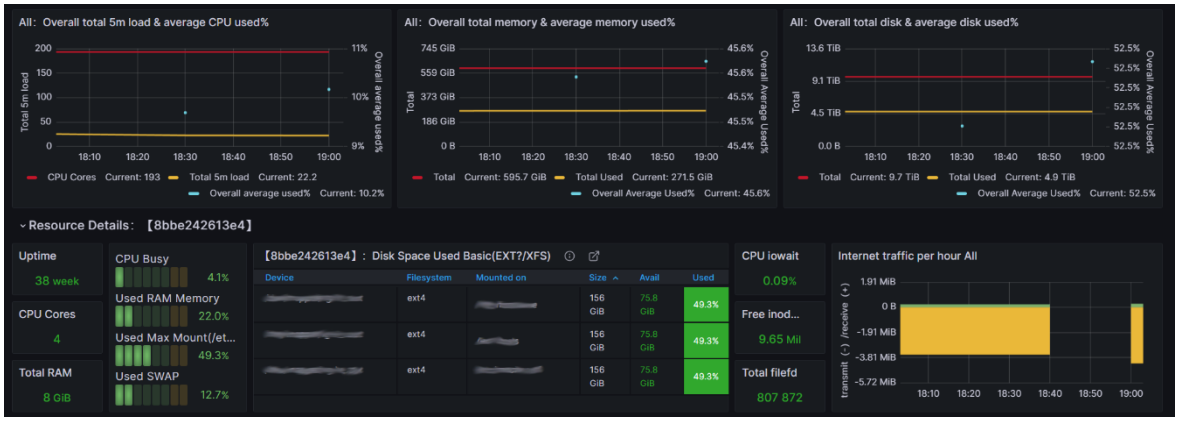
Zde se vizualizace bude týkat několika navržených řešení na základě odlišných use cases. V rámci použití architektury s nástrojem Prometheus to budou konkrétně:

- Přehled všech serverů a jejich resources
- Disk usage – s alerty
- Přehled CPU a RAM Docker containerů určitých serverů

Vzhledem ke stejné architektuře, bude zde u všech panelů konkrétních dashboardů použit jazyk PromQL přímo související s nástrojem Prometheus. Jak již bylo zmíněno v kapitole výše, monitoring přehledu serverů již existoval na instanci Grafany sabo.gmbh.de. Avšak nebyl moc efektivní, proto je tedy vytvořen nový dashboard, také inspirovaný předdefinovanými Grafana Labs dashboardy konkrétně s ID dashboardu 15172. [86] Dashboard však bude nasazen na novou Grafanu devops monitoring. Vzhledem k rozsahu práce budou zde vyobrazeny pouze první dva snímky dashboardu a zbylé dva v přílohách viz. *příloha E – F.* Vizualizace první poloviny dashboardu tedy vypadá následovně:




*Obrázek 9 – Dashboard přehledu všech serverů 1/4*



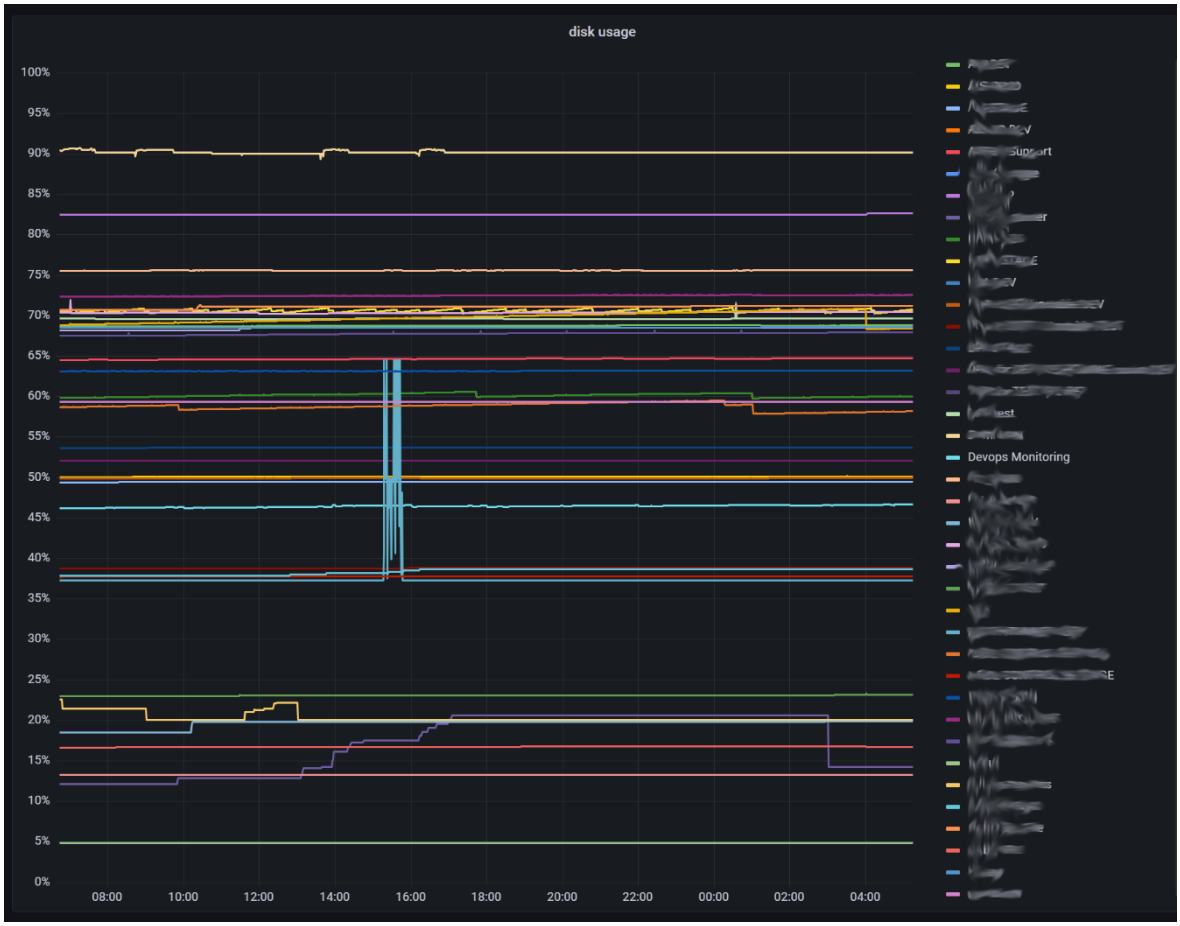

*Obrázek 10 – Dashboard přehledu všech serverů 2/4*

Tyto čtyři obrázky vyobrazují celý dashboard týkající se využití resourcu všech serverů. Nejprve viz. *Obrázek 9* Je viditelný celý úvod. Nejprve vpravo nahoře je možnost výběru time range a reloadu. Time range určuje dobu, po kterou se mají data zobrazovat jako například poslední hodina, která byla právě vybrána pro toto zobrazení a u reloadu je možné nastavit časový interval, po kterém se má stránka reloadovat a tím načítat nová data, zde byl vybrán automatický reloadu každých 15 sekund. Hned pod tímto úvodem je již zobrazení samotného dashboardu. Nejprve zobrazení jeho proměnných jakožto možnosti filtrace, kde je nejzásadnější zejména JOB a Instance. Zde je možno filtrovat skupiny serverů dle jejich job names nebo přímo konkrétní servery. Pro vizualizaci v této práci byl vybrán pouze server

devops monitoring. Následně je zde vyobrazen již první panel, a to konkrétně tabulka, která je velice rozsáhlá. Na tuto tabulku totiž filtrace nemá vliv, což opět vede k nutné cenzuře všech ostatních názvů serverů. Je zde možné vidět přehled prakticky všech důležitých hodnot jako je procentuální využití CPU, jeho load a počet jader, RAM, disku včetně read a write, uptime nebo taky download a upload networku. Jedná se tedy o velice efektivní panel, včetně barevně odlišených tresholdů zelená, oranžová a červená u přehledů procentuálních využití, což poskytuje opravdu dokonalý přehled na první pohled. I z tohoto důvodu je právě celý tento panel také nonstop zobrazen na televizi v kanceláři, kde se každou minutu posouvá a průběžně tak vyobrazí status všech serverů spravovaných firmou. Tento panel je přehledný a směrodatný nejen pro pracovníky DevOps, ale také všechny ostatní zaměstnance jako například vývojářů, kterých se určité servery mohou také týkat. Zbytek dashboardu viz. *příloha E - F* zobrazuje převážně time series specifičtější grafy, určené převážně zaměstnancům DevOps. Mimo time series grafů se zde však také nachází zajímavá tabulka s přehledem jednotek disku a jejich využití, tři panely typu stats nabízející pohled na kapacitu serveru.

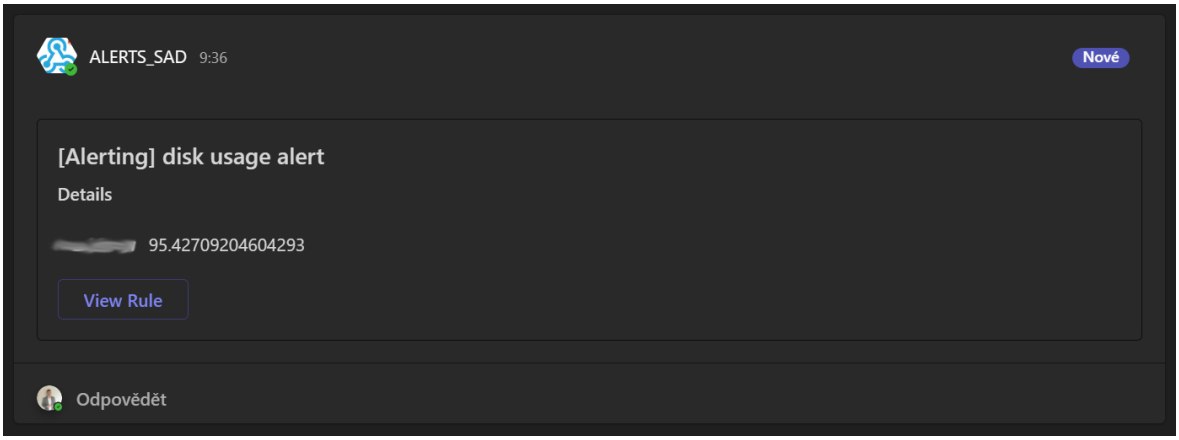
Dalším dashboardem architektury využívající několik stejných metrik, které využívá i předešlý dashboard je disk usage. Tento dashboard pouze s jedním panelem slouží převážně pro možný alerting do firemního komunikačního kanálu Teams. Mimo to poskytuje také jasný přehled nad využitím disku napříč všemi servery, neboť právě disk usage je jedním z nejkritičtějších prostředků, u kterého na některých serverech často dochází k jeho zaplnění a je tak nutné jeho předcházení. Oproti předešlému dashboardu je celý tento dashboard dělaný speciálně bez předlohy z Grafana Labs. Pomocí PromQL je zde panel nadefinovaný následovně:

```
max((node_filesystem_size_bytes{fstype=~"ext.?|xfs"}-
node_filesystem_free_bytes{fstype=~"ext.?|xfs"})
*100/(node_filesystem_avail_bytes
{fstype=~"ext.?|xfs"}+(node_filesystem_size_bytes{fstype=~"ext.?|xfs"}-
node_filesystem_free_bytes{fstype=~"ext.?|xfs"})))by(instance)
```




*Obrázek 11 – Dashboard přehledu využití disku*

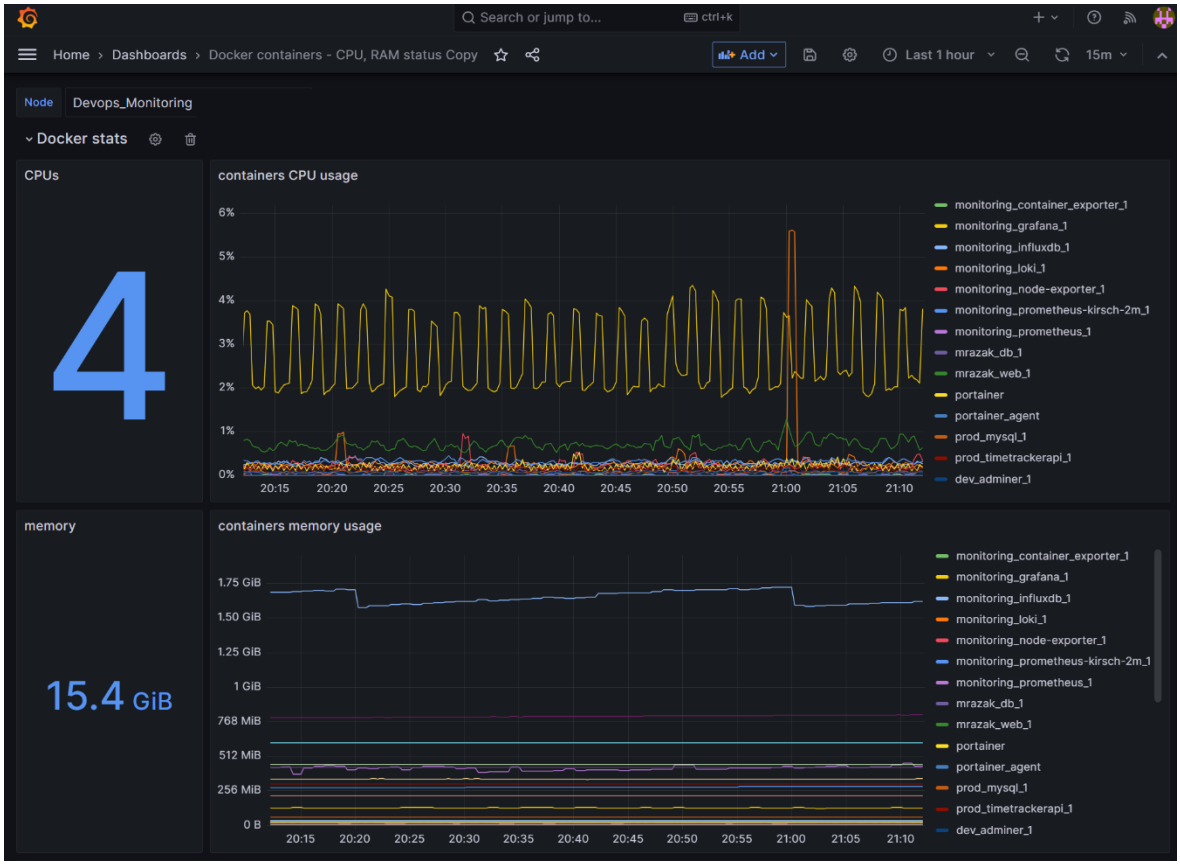
Ze samotného panelu je vidět, že spravované servery pokrývají téměř celou škálu rozsahu od 0% do 100%. Hodnota je zde zvolena právě procentuální vzhledem k různým kapacitám disku každého ze serverů. Následně je zde tedy vytvořený alerting, notifikující každý server, který se dostane přes 95% využití disku. Samotný setup je pak vyobrazen v *příloze G.* Samotný alert je pak zachytávaný rozšířením v Teams o aplikaci Incoming Webhook a v Teams vypadá následovně:




*Obrázek 12 – Dashboard přehledu využití disku – ukázka alertu*

Je zde tedy zobrazen název alertu, o který se jedná, název serveru, který má potíž s diskem a jeho aktuální hodnota, která však v době alertingu bude vždy kolem hodnoty 95%. Po odstranění problému chodí také notifikace se stavem [OK] namísto stavu [Alerting].

Poslední vizualizací v rámci této architektury je tedy přehled využití containerů, kde je vyžadován pouze přehled využití CPU a RAM. Toto řešení jako první vyžaduje container exporter místo původního node exporteru. Tento dashboard byl opět vytvářen zcela samostatně bez využití templatu. Jak již bylo zmíněno, zaměřuje se a je tedy nasazen pouze na pár serverech kde je monitoring containerů vyžadován. Byl však přidán i na server devops monitoring pro možnou lepší prezentaci dashboardu s minimální cenzurou. Dashboard vypadá následovně:




*Obrázek 13 – Dashboard přehledu Docker containerů*

Je zde tedy opět možná filtrace pro výběr jednoho nebo více serverů, jejichž Docker containery chceme zobrazit. Následně jsou vytvořeny 4 panely, 2 týkající se CPU a 2 týkající se RAM. Vždy jeden v rámci zobrazení kapacity serveru, ať už jako počet jader v případě CPU tak maximální kapacita RAM v GB. Vedle těchto panelů typu stats jsou s nimi související time series grafy. CPU udávané v % a RAM v MB / GB. Nadefinování těchto panelů pomocí PromQL pak vypadá následovně.

Pro containers CPU usage:

```
sort_desc(sum(rate(container_cpu_usage_seconds_total{name=~".*",instance=~"($Node).*"}[1m])) by (instance, name))
```

Pro containers memory usage:

```
sort_desc(sum(container_memory_usage_bytes{name=~".*",instance=~"^($Node).*"}) by (instance, name))
```

### 4.4.3 Nasazení nástroje InfluxDB

Zde se tedy jak již bylo zmíněno jedná o time series databázi, kam je možné efektivně a pravidelně zasílat data ze serverů. Stejně jako předešlé nástroje je i InfluxDB nasazen v *docker-compose.yml* viz.:

```
influxdb:
  image: influxdb:latest
  restart: always
  volumes:
    - ./influx/tempo:/tempo:rw
    - ./influx/influxdb2:/var/lib/influxdb2:rw
  ports:
    - 127.0.0.1:<PORT>:8086
```

Zde byl tedy použit image *influxdb:latest* s nadefinováním tagu verze latest což představovalo v době nasazení verzi v2.7.1. Jak je následně vidět i ve volumes, jedná se tedy o druhou generaci InfluxDB. V rámci volumes zde bylo však nutné vytvořit nejprve dočasný adresář tempo, pro stejný případ jako v případě Grafany, pouze zde byla použita mírně odlišná metodika. Opět se však jedná o prvotní vykopírování celého repositáře z containeru na server, aby následně mohl být vytvořen celý repositář nakopírován při opakovaném spuštění ze serveru do containeru zpět. To tedy zajistí zachování celé struktury při stavu containeru down například během upgradu. Zde následná konfigurace samotné databáze proběhla přímo v UI nasazeného containeru.

Ten byl vystaven z localhostu ven z localhostu na DNS influx.devops-monitoring.com opět skrze Nginx server viz. *příloha H.* V té se opakují opět stejné prvky, pouze s jednou základní location / a navíc zde nebylo potřeba ani použití doplňujících údajů pro tento nástroj.

Samotné UI pak vypadá konkrétně v sekci Buckets, pro tuto práci nejzásadnější viz. *příloha I.* Zde je navíc možné i vytváření a použití více organizací pro odlišení prostředí například jako v rámci této práce dle přiřazených klientů k daným řešením. Zde, dle již zmíněných požadovaných řešení to jsou právě organizace Sabo, zastupující interní projekty firmy, Comtac a Kirsch což je možné vidět v *příloha J.*

### 4.4.4 Customizovaný monitoring containerů

Jak bylo popsáno v kapitole výše, klasický monitoring využití prostředků, konkrétně CPU a RAM Docker containerů je již nasazen pomocí nástroje Prometheus, avšak zde vznikla ještě potřeba od klienta Kirsch na monitorování uptimu vzhledem ke stavům up / down. Pro tento use case však container exportery s jejich předdefinovanými metrikami nedostačující. Pro tento účel bylo vytvořeno řešení viz. níže.

#### 4.4.4.1 Bash scripty pro získávání dat

Nejen pro toto řešení vznikla tedy potřeba vytváření vlastních scriptů pro vytahování dat, konkrétně scriptů v Linuxovém prostředí bash. Zde byly vytvořeny 2 scripty pro každé z prostředí Kirsch (DEV, TEST, STAGE, PROD) serverů. Tyto scripty se mezi prostředími odlišují pouze malými detaily. První script je prezentován na příkladu prostředí DEV viz. *příloha K.* Ten se stará právě o získávání dat z Docker containerů v podobě hodnot 1 = up nebo 0 = down. Jelikož je toto prostředí vytvářeno v prostředí bash, je nutné zde zároveň vytvářet pomocné soubory, těch se v tomto řešení vyskytuje 7. V prvním scriptu je tedy nejprve uvedeno obecné definováni bash scriptů *#!/bin/bash* stejně jako u všech následujících scriptů. Následně je zde definován timestamp pro přehled o čase přiřazenému k aktuálním hodnotám a zároveň pro možnost odesílání do InfluxDB. Dále je zde poměrně složitý proces přepisování hodnot na základě výpisu příkazu *docker stats*, kde jsou ve stručnosti nacházena klíčová slova a ty jsou následně přepisovány do již zmíněných hodnot 1 nebo 0. Druhý script viz. *příloha L* je spouštěn prvním scriptem a stará se o samotné zasílání dat do InfluxDB. Zde byla nutnost použití druhého scriptu, neboť počet Docker containerů může být variabilní a v rámci jednoho scriptu nebylo možné vytvořit curl request s počtem proměnných odpovídajícím počtu containerů. Tímto způsobem je zajištěno to, že script vždy pozná, kolik containerů se na serverů nachází a přesně tolik jich i pošle do InfluxDB. Konkrétně pak tato struktura scriptů odesílá do InfluxDB následující řadky dle počtu containerů:

```
containerStatus,name='$ps_1' dev='$status_1' '1710023101'000000000
```

Zde *containerStatus* definuje název metriky, následně je zde definován skrze proměnnou název containeru a jeho hodnota pod názvem *dev*, dle prostředí serveru. Poslední

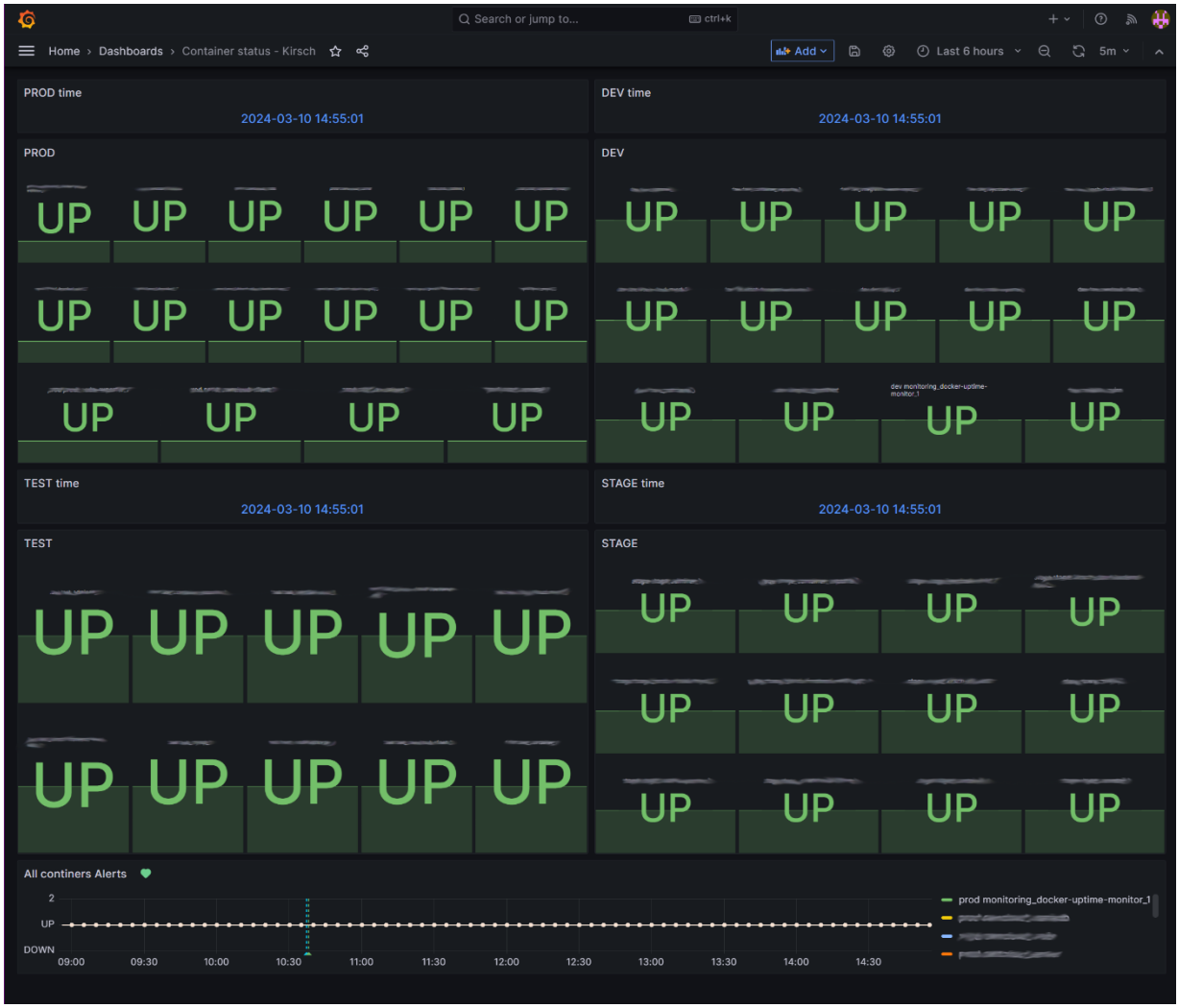
sloupec vyobrazuje již pouze timestamp udávající aktuální čas, který je definován také proměnnou z předchozího scriptu.

#### 4.4.4.2 Nastavení bucketů

V nástroji InfluxDB je nutné pro možnost zasílání dat vytvoření tzv. bucketů. Ty představují základní stavební bloky pro ukládání dat jakožto organizovaný prostor pro ukládání časových řad, tzv. timeseries dat. Pro každé řešení tedy vzniká potřeba vytvoření vlastního bucketu s vlastním názvem a politikou uchovávání dat, tzv. retention policy. Pro toto řešení byla v InfluxDB vytvořena organizace Kirsch a v této organizaci následně konkrétní buckety přidělené daným prostředím viz. *příloha M* na které je možné zároveň vidět, že pro toto řešení bylo dostačující použití 30. denní retention policy pro každý z bucketů. Pro tyto buckety byla zároveň vytvořena konvence názvů, kde se nejprve vyskytuje předpona názvu prostředí a následně container status. V některých případech jsou tyto buckety vytvořeny 2x s koncovkou 2. V takových případech bylo nutno po nasazení řešení provést určité změny což vedlo k vytvoření nového bucketu se zachováním původních jakožto backupů. *Příloha N* pak ukazuje na vizualizaci samotného bucketu ukázanou opět na příkladu prostředí DEV. Je možné zde vidět sloupce obsahující hodnoty nebo názvy vycházející právě ze scriptů popsaných v předchozí kapitole. Názvy těchto bucketů následně slouží k nadefinování panelů v Grafaně pomocí jazyka Flux.

#### 4.4.4.3 Vizualizace v Grafaně

Pro vizualizaci dashboardu byl do Grafany naimportován InfluxDB a následně vytvořen dashboard s názvem Container status – Kirsch. Vytvořený dashboard vypádá následovně:

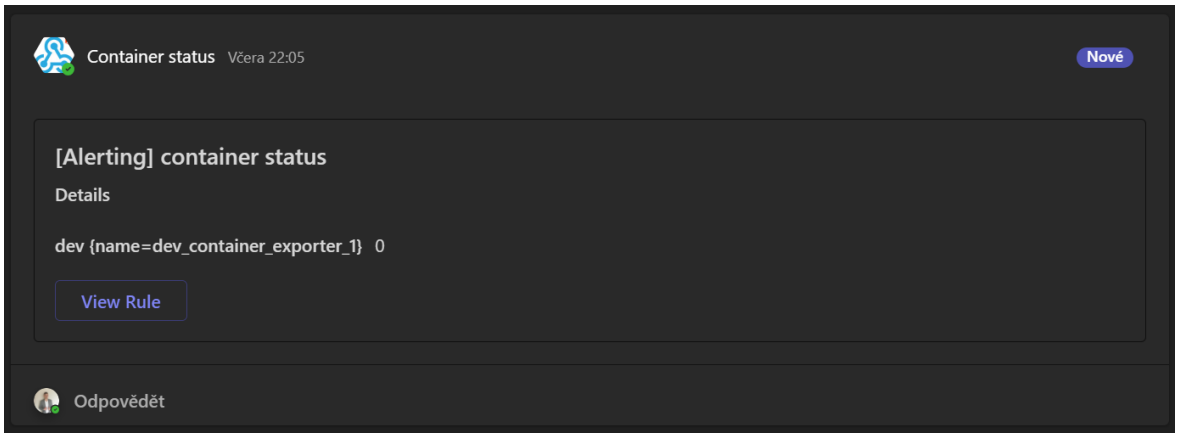



*Obrázek 14 – Dashboard přehledu container uptime*

Zde jsou tedy viditelné 4 panely rozdělené dle daných prostředí a každý z nich zobrazuje uptime všech containerů na daném serveru. Jak je i z dashboardu patrné, každé prostředí může obsahovat jiný počet containerů což je také důvod, proč byla výše nutnost upravení scriptů na dva vzhledem k variabilním počtům Docker containerů. Nadefinování query pro tyto panely pomocí jazyka Flux vypadá opět na příkladu prostředí DEV následovně:

```
from(bucket: "dev_container_status2")
    |> range(start: -1d)
    |> filter(fn: (r) => r["_field"] == "dev")
```

Dále se zde nachází nad každým z těchto panelů přehledu containerů panely také ke každému prostředí s uvedeným časem poslednímu updatu stavu uptimů podle timestamp ze scriptu. Dále se zde pod těmito 8 panely nachází jeden panel se stavem všech containerů dohromady na kterém je nastavený alerting opět do interního komunikačního kanálu Teams, avšak s rozdílným cílovým kanálem přímo určeným klientovi Kirsch. Tyto aletry jsou opět v Teams zachytávány pomocí služby Incoming Webhook. Nastavení alertingu v Grafaně je vyobrazeno v *příloze O* a samotný alerting pak v kanálu Teams může vypadat například následovně:




*Obrázek 15 – Dashboard přehledu container uptime – ukázka alertu*

### 4.4.5 Monitoring portů

V rámci tohoto řešení tedy vznikla potřeba vycházející od klienta Comtac ověřovat uptime nikoli Docker containerů, jak bylo v předchozím řešení, ale portů samotných aplikací nasazených v prostředí u klienta. Pro toto řešení byl tedy vytvořen jeden script se třemi pomocnými soubory. Každý z těchto souborů reprezentuje stav daného portu, neboť porty, které mají být sledovány jsou právě 3. Tyto porty však musí být vzhledem k zásadám cenzurovány.

#### 4.4.5.1 Bash scripty pro získávání dat

V *příloze P* je tedy vyobrazený script použitý pro toto řešení. Avšak z důvodu velkého počtu vyskytování portů ať už jako součástí příkazů, nebo jako názvů proměnných. Včetně IP adresy Comtac serveru toho zde musí být dost cenzurováno. Tento script však

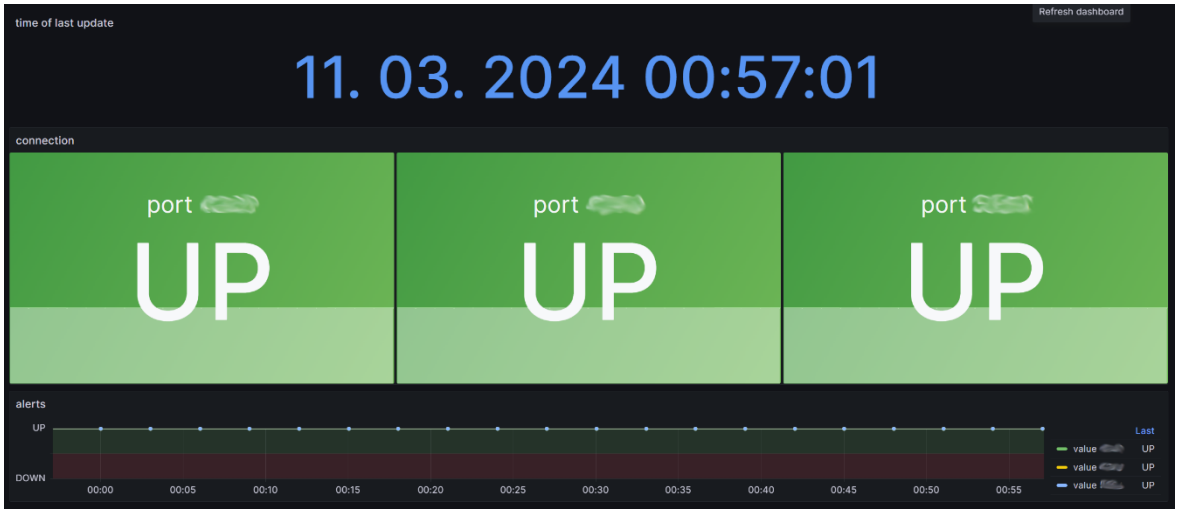
nejprve zjišťuje příkazem telnet dostupnost daného portu, kterou zapisuje do pomocného souboru. Následně jsou pak výpisy z příkazu telnet opět převedeny na hodnoty 1 (UP) / 0 (DOWN). A tyto hodnoty jsou následně opět zasílány do nástroje InfluxDB. Avšak tento script kromě pouhého odesílání monitorovacích hodnot také spouští vzdáleně druhý script na serveru klienta Comtac, který danou službu restartuje. Tento script však není součástí práce.

#### 4.4.5.2 Nastavení bucketů

Pro toto řešení byla v nástroji InfluxDB vytvořena organizace s názvem comtac obsahující bucket s názvem connection. Tento bucket je zobrazen v *příloze Q*. Zde jsou vidět pouze tři sledované hodnoty, podle právě třech portů.

#### 4.4.5.3 Vizualizace v Grafaně

Následně byl vytvořen dashboard viz:




*Obrázek 16 – Dashboard přehledu uptime portů*

Tento dashboard opět zobrazuje na prvním místě panel času posledních zobrazených dat. Pod ním jsou zobrazeny přepsané hodnoty z 1 / 0 na UP / DOWN, přiřazené každému z portů. Tento panel je nadefinován pomocí Flux query úplně jednoduše:

```
from(bucket: "connection")
    |> range(start: -1d)
```

Potřebné přepisy a úpravy tohoto panelu se odehrávají v samotném UI grafany týkající se nastavení panelů. Pod tímto ponelem se nachází už jen poslední panel s grafem, který se stará o alerting. Zde však není možné výsledný alerting ukázat, neboť za poslední měřenou dobu žadný ze sledovaných portů nespadnul do stavu DOWN.

### 4.4.6 Monitoring SSL certifikátů

Posledním řešením v rámci využití nástroje InfluxDB a bash scriptů je monitoring expirací SSL certifikátů s využitím alertingu několik dní předem. Jak již bylo zmíněno, firma Sabo spravuje opravdu veliké množství DNS s využitím SSL certifikátů a tento počet je zároveň velice variabilní. Proto se zde musí počítat s robustním řešením, které je schopné zvládat veliký počet hodnot v proměnných a fakt, že je tento počet variabilní.

#### 4.4.6.1 Bash scripty pro získávání dat

Dané řešení zde představuje jeden bash script a jeden soubor s poolem DNS, kde se bude ověřovat expirace jejich SSL certifikátů. Script pro sběr dat je vyobrazen s drobnou cenzurou citlivých údajů a se zkráceným rozsahem řádků pro každý z certifikátu (neboť těch je kolem 200) v *příloze R.* V tomto scriptu se na základě výpisu z příkazu *curl https://$domain_space -vI --stderr* – zjišťuje po dosazovaných hodnotách DNS ze souboru *.env-domains* čas, který zbývá do jejich expirace. Avšak výpis předešlého příkazu zobrazuje obecný zápis datumu, s kterým není možné dále pracovat. Proto je zde následně tento výpis přepracován do hodnoty timestamp. Na základě toho jsou pak kažné DNS přiřazovány hodnoty 0-7, při čemž hodnota 7 představuje neznámou chybu typu error (může nastat například v důvodu network traffic, nebo již zrušené DNS). Hodnota 0 naopak znamená, že DNS má do expirace víc jak 10 dnů a je tedy ve stavu OK. Ostatní hodnoty 1-6 představují čas pro možný alerting z Grafany, při čemž hodnota 1 = 10 dnů do expirace, hodnota 2 = 5 dnů do expirace, hodnota 3 = 3 dny do expirace, hodnota 4 = 2 dny do expirace, hodnota 5 = 1 den do expirace a hodnota 6 znamená, že SSL certifikát již exspiroval. Následně je zde nadefinováno spoustu řádků pro curl request do bucketu InfluxDB:

```
$measurement,${field}=${domain_1} ${value}=${status_1} ${timestamp}000000000
```

Tyto řádky obsahují skrze proměnné na prvním místě název metriky, poté název DNS, o kterou se jedná, dále dané DNS přiřazenou hodnotu stavu expirace 1-7 a na závěr hodnotu aktuálního času tzv. timestamp.

#### 4.4.6.2 Nastavení bucketů

Pro tyto hodnoty z předešlé kapitoly je v InfluxDB vytvořen bucket pod organizací Sabo s názvem SSL_Certs a retention policy nastavenou na 30 dnů viz. *příloha S.* Zde jsou vidět cenzurované DNS a k nim přiřazené denní hodnoty. U tohoto řešení je dostačující odesílání dat pouze 1x denně, což je však detailněji popsáno v kapitole níže.

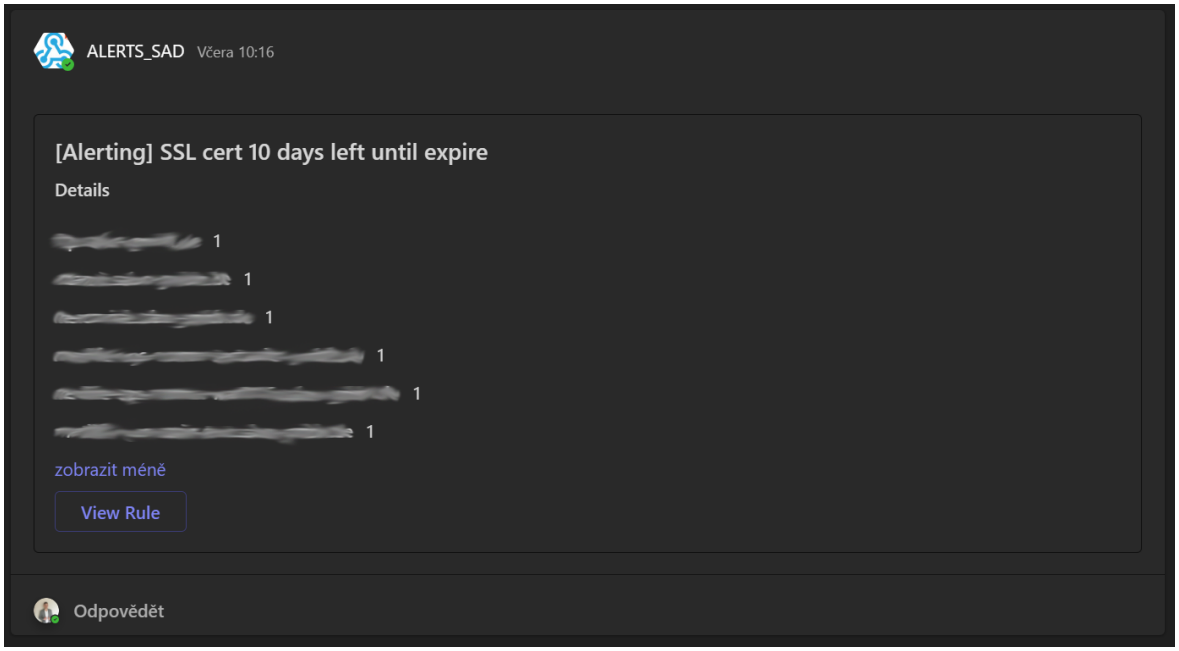
#### 4.4.6.3 Vizualizace v Grafaně

Pro toto řešení byl vytvořen opět vlastní dashboard bez použití templatů viz:


*Obrázek 17 – Dashboard expirací SSL certifikátů*

Z *Obrázku 17* je možné vidět dashboard nadefinovaný opět pomocí query Flux viz. *příloha T.* Dále je z tohoto dashboardu vidět, že hodnoty 0-7 byly kvalifikovány a přepsány do významu, který reprezentují. Zároveň je samotný graf odlišen barevnými tresholdy. Z daného příkladu je možné vidět spodní linku ve stavu OK představující DNS, u kterých mají SSL certifikáty do expirace déle než 10 dnů. Na opačné straně ve stavu ERROR je zase linka DNS, které nejsou pravidelně dostupné pro curl dotaz, což znamená, že jsou

pravděpodobně mimo službu. Avšak k těmto DNS ve stavu ERROR 2x vyskočila na jeden den modrá linka. To zase značí DNS pravděpodobně ve stavu OK, později 10 days, avšak do stavu ERROR se dostala pravděpodobně pouze díky vysokému využití sítě v moment spuštění monitorovacího scriptu. Následně je zde vidět žlutá křivka skupiny DNS, které se již přesunuly do stavu značící 10 nebo méně dnů do expirace. Zároveň je zde vidět červená, svislá, přerušovaná čára, která značí, že v moment, kdy DNS dostaly do stavu 10 days byl odeslán alert opět do kanálu Teams. Tyto aletry v kanálu Teams vypadají následovně:




*Obrázek 18 – Dashboard expirací SSL certifikátů – ukázka alertu*

### 4.4.7 Nastavení Cron jobů

Pro všechna řešení, v této práci popsána od kapitoly *4.4.4 Customizovaný monitoring containerů* až k této kapitole bylo potřeba vytvoření také cron jobů, které umožňují dle příkazů podle cron syntaxe spouštět vytvořené bash scripty pro monitorovací řešení v předdefinovaných časových intervalech. Zde však vzniká potřeba vytvoření různých časových intervalů pro scripty přiřazené k daným řešením.

Co se samotné syntaxe týče, cron job se skládá z nadefinování časového intervalu následovaný příkazem, v tomto případě příkaz pro spuštění scriptu. Pro nadefinování

časového intervalu je zde 5 pozic. Kde od první k poslední pozici jsou následující hodnoty: minuta, hodina, den v měsíci, měsíc, den v týdnu. Pro vynechání nadefinování některé z pozic se používá znak *. Následující 3 řádky znázorňují cron joby pro každé z řešení uvedených výše:

```
*/5 * * * * cd /root/status-container && bash status.sh
*/3 * * * * bash /root/scripts/comtac_ems/check-ems-port.sh
0 6 * * * bash /root/scripts/ssl_expiration/expiration.sh
```

Zde se první řádek týká serveru klienta Kirsch, kde je nasazeno řešení zajišťující monitoring a alerting Docker container uptime. Vyskytují se zde sice 2 scripty, a však pro nadefinování v cronu stačí uvést jen jeden, který následně spouští i druhý script. Zde byl nastaven časový interval pro spouštění každých 5 minut. Další dva řádky se již týkají serveru devops monitoring, kde je první nadefinován script ověřující status portů spouštějící se každé 3 minuty a druhý je nadefinován script ověřující expirace SSL certifikátů, spouštějící se 1x denně, přesně v 6:00 ráno.

### 4.4.8 Monitoring logů aplikací

Další architektura, kterou je potřeba naimplementovat je tedy vytahování aplikačních logů ze serveru do Grafany tak, aby k nim měli vývojáři přístup a mohli tak snáze debuggovat své aplikace. Pro tento use case je tedy implementován nástroj Loki, který však oproti předchozím nenabízí své UI, pouze zpracovává a zprostředkovává získané logy ze serverů do Grafany. Tyto data v podobě logů jsou ze serverů získávány pomocí nástroje Promtail, nebo JSON scriptů. Součástí této práce je však pouze nasazení a využití architektury využívající nástroj Promtail.

#### 4.4.8.1 Nasazení nástroje Loki

Pro toto řešení je tedy nejprve nutné nasazení nástroje Loki. Tento nástroj je opět nasazen ve stejném *docker-compose.yml* na serveru devops monitoring jako předešlé nástroje. Jeho konfigurace zde vypadá následovně:

```
loki:
  image: grafana/loki:2.4.0
  user: root
  restart: always
  volumes:
    - ./loki/local-config.yaml:/etc/loki/local-config.yaml
  ports:
    - 127.0.0.1:<PORT>:3100
  command: -config.file=/etc/loki/local-config.yaml
```

Zde byla použita konkrétní verze viz image *grafana/loki:2.4.0*, jelikož zde se tato verze osvědčila již při předešlém nasazení firmou. Následně zde bylo nutné nadefinování uživatele na úrovni administrátor *root*. Následně je zde důležité nadefinování volumes, kde se do containeru kopíruje konfigurační soubor nástroje Loki s názvem *local-config.yaml*. Tento soubor je následně spouštěn v sekci command viz příkaz *-config.file=/etc/loki/local-config.yaml*. Tento konfigurační soubor je vyobrazen v *příloze U.*

#### 4.4.8.2 Nasazení nástroje Promtail

Dalším nástrojem, který je třeba nasadit je tedy Promtail. Ten však již není nutné nasazovat ne serveru devops monitoring, ale na serverech, kde se vyskytují aplikace jejichž aplikační logy chceme sledovat. V případě této práce se tedy jedná o servery klienta Kirsch. V praxi pak bylo později přidáno ještě více klientů, kteří si vyžádali vizualizaci aplikačních logů v Grafaně. Zde samotné nasazení v Dockeru pomocí souboru *docker-compose.yml* vypadá následovně:

```
promtail:
  restart: always
  image: grafana/promtail:1.6.1
  volumes:
    - ./promtail/config.yml:/etc/promtail/config.yml
    - ./promtail/logs:/logs
  command: -config.file=/etc/promtail/config.yml
```

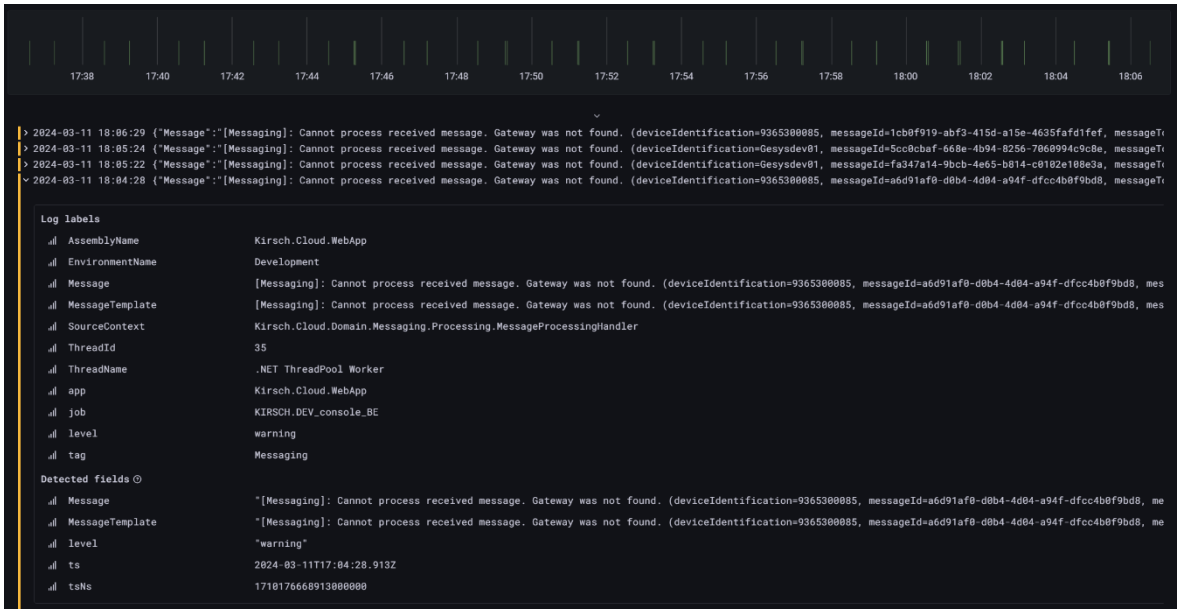
U nástroje Promtail, který je tedy také stejně jako Loki bez UI, byla použita osvědčená verze pro nasazení viz. image *grafana/promtail:1.6.1*. Zde jsou opět velice důležité volumes, které se skládají ze dvou částí. První se stará o nakopírování konfiguračního souboru s názvem *config.yml* a druhá o přenesení samotných logů do Docker

containeru. Následně je zde také nadefinován příkaz pro spuštění konfiguračního souboru *-config.file=/etc/promtail/config.yml*. Zde tedy opět na příkladu prostředí DEV klienta Kirsch vypadá konfigurační soubor viz. *příloha V.* Tato konfigurace je již oproti konfiguraci nástroje Loki podstatně zajímavější. Definuje totiž cesty logů a názvy samotných instancí aplikačních logů v *job_name*. Pro nadefinování cest k logům je však potřeba se také domluvit s vývojáři, neboť právě ti musí v rámci jejich aplikace vytvořit soubor, kam bude aplikace vypisovat své logy. Tento soubor je následně skrze volumes přenesen do containeru nástroje Promtail. Jak již bylo zmíněno, touto metodou následně Promtail přeměňuje loglines na logfiles. Tím oproti řešení za použití JSON scriptu šetří využití disku serveru, kde je nasazen nástroj Loki, tedy devops monitoring. Avšak znemožní efektivní procházení logů typu multilines.

#### 4.4.8.3 Vizualizace v Grafaně

Jak již bylo nastíněno, zde se bude vizualizace rozdělovat podle použité architektury. Konkrétně budou odlišné parsery v nadefinování query panelů logů podle toho, jestli se bude jednat o architekturu s použitím JSON scriptů nebo nástroje Promtail.


*Obrázek 19 – Dashboard aplikačních logů – Promtail architektura*




*Obrázek 20 – Dashboard aplikačních logů – JSON architektura*

Na *Obrázku 19* je možné vidět vizualizaci architektury za použití nástroje Promtail, který byl nasazen v minulé kapitole. Tento Dashboard byl inspirován předdefinovaným dashboardem z Grafana Labs [87]. Na horním grafovém panelu je možné vidět počet obdržených jednotek logů za daný časový úsek. Pod ním je již samotný výpis aplikačních logů. Avšak je možné vidět neefektivní zpracování multilinest logů, které pokračují na novém řádku, přestože náleží k předešlým řádkům. Oproti tomu na *Obrázku 20* je možné vidět vizualizaci za použití architektury s JSON scripty. Zde je možné vidět, že všechny logy ať jsou jakkoli dlouhé pokračují stále na jednom řádku. Tím je také možné rozbalit nabídku každého z řádků s přehledem všech hodnot a proměnných přiřazených právě k těmto řádkům. Rozdíl mezi nadefinovánými query v těchto panelech pomocí jazyka LogQL je pak následovný:

```
{job="$app"} |= "$search" | logfmt
```

```
{job="$app"} |= "$search" | json
```

První řádek vyobrazuje LogQL s parserem *logfmt* pro architekturu používající nástroj Promtail, zatímco druhý řádek využívá parser *json*.

### 4.4.9 Monitoring logů containerů a jejich správa

Toto řešení již bylo v rámci monitoringu logů containerů částečně vyřešeno v předchozí kapitole *4.4.8 Monitoring logů aplikací*, avšak toto řešení za použití nástroje Portainer umožňuje vývojářům kompletní přístup k jejich Docker containerům v rámci samotného UI nástroje Portainer. Tento nástroj mimo výpisů container logů (bez jakékoli změny a nutnosti vytvoření souboru s logy v rámci aplikace) nabízí také obecnou správu těchto Docker containerů.

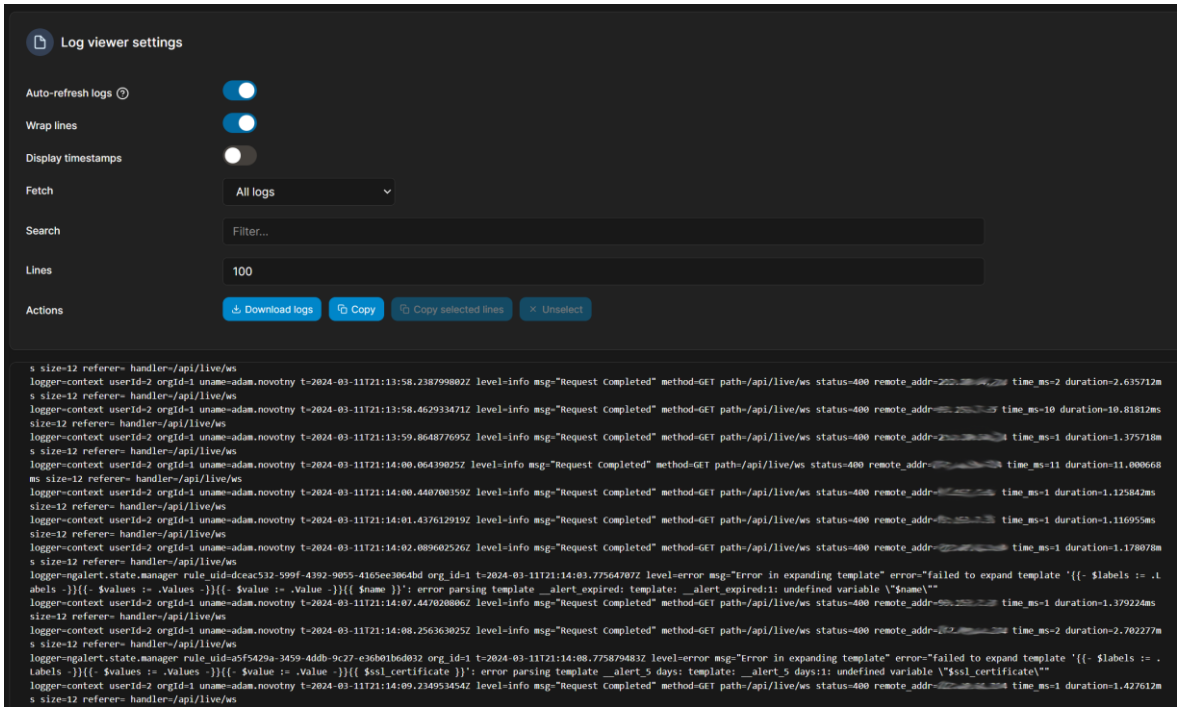
#### 4.4.9.1 Nasazení nástroje Portainer

Pro toto řešení je tedy nasazen opět prostřednictvím *docker-compose.yml* následující setup nástroje Portainer:

```
portainer:
  image: portainer/portainer-ce:latest
  container_name: portainer
  restart: always
  volumes:
    - /var/run/docker.sock:/var/run/docker.sock
    - portainer_data:/data
  ports:
    - <PORT>:8000
    - 127.0.0.1:<PORT>:9000
```

Zde byl použit image s tagem verze latest *portainer/portainer-ce:latest*. Co se volumes týče, první z nich obsahuje stejný volumes co portainer agenti, neboť vytahuje informace o Docker containerech svého hostovaného serveru a není tak již potřeba na daný server nasazovat Portainer agent. Druhý zprostředkovává zálohu dat například v případě upgradu. Z portů je zde vystavený ven druhý, který je zároveň uzavřen na adresu localhost. Ven se tedy dostane přes následnou konfiguraci Nginx viz. *příloha W*. Z této konfigurace je vidět, že Portainer jako jediný byl nasazen na DNS firmy portainer.sabo-gmbh.de, neboť je jeho přístup zprostředkovávaný několika klientům, kteří nejsou obeznámeni se strukturou serveru devops monitoring. Dále je zde vidět že pro bezchybný chod bylo za potřebí použití několika CORS a nově také *client_max_body_size 200M;* který udává velikost 200M pro možné stahování například textových souborů v podobě logů přímo z UI Portaineru. Samotné UI úvodní strany přehledu serverů je zobrazeno v *příloze X.* Z té je zároveň možné

vidět, že v rámci monitoringu Portainer nenabízí striktně pouze hodnoty týkající se Docker containerů, ale například také přehled CPU jader a velikosti RAM samotného serveru. Po výběru serveru, je možné vidět UI přehledu Docker containerů nasazených na vybraném serveru viz. *příloha Y.* Pro tuto ukázku byl vybrán právě server devops monitoring. Z tohoto přehledu je zas možné vidět detailní přehled všech nasazených Docker containerů, včetně možnosti spuštění console Docker containeru přímo zde, v UI Portaineru. Nebo právě již zmiňované Docker container logy:




*Obrázek 21 – Portainer UI – logy Docker containeru*

#### 4.4.9.2 Nasazení agentů

Pro toto řešení je však také nutné nasazení na sledované servery Portainer agentů. Ti jsou opět nasazovány prostřednictvím *docker-compose.yml*:

```
portainer:
  image: portainer/agent:2.9.3
  container_name: portainer_agent
  restart: always
  volumes:
    - /var/lib/docker/volumes:/var/lib/docker/volumes
    - /var/run/docker.sock:/var/run/docker.sock
  ports:
    - <PORT>:9001
```

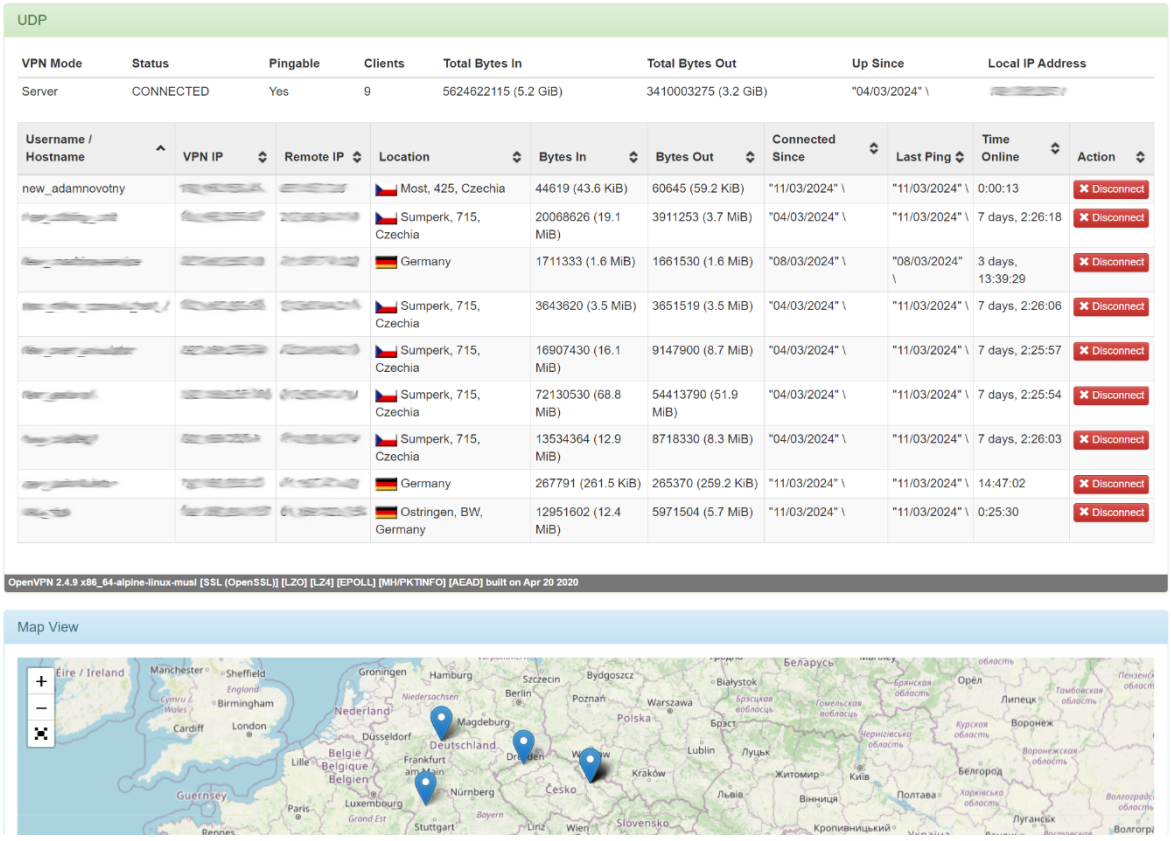
Zde je tedy použit image s konkrétní verzí agenta *portainer/agent:2.9.3*. Oba řadky volumes se v tomto případě starají o kopírování dat právě sledovaných Docker containerů směrem ze serveru do Portainer agenta, ale i obráceně. Neboť Portainer neslouží pouze jako vizualizace získaných dat, ale je zde možno také provádět změny v Docker containerech. Port je zde následně odemčený bez localhostu, neboť agenti nejsou publikováni skrze Nginx a musí být viditelné ze serveru, kde je Portainer nasazen.

### 4.4.10 Nasazení VPN monitoringu

Řešení monitoringu aktivních VPN certifikátů je stejně jako samotná služba OpenVPN nasazena v Dockeru na stejném serveru. To je nezbytné pro funkčnost daného řešení. OVPN monitor, který je zde použit pro monitoring je nasazen v docker-compoze.yml viz.:

```
openvpn-monitor:
  image: ruimarinho/openvpn-monitor
  container_name: openvpn-monitor
  ports:
    - 127.0.0.1:<PORT>:80
  restart: always
  environment:
    - OPENVPNMONITOR_DEFAULT_DATETIMEFORMAT="%%d/%%m/%%Y" \
    - OPENVPNMONITOR_DEFAULT_LATITUDE=-37
    - OPENVPNMONITOR_DEFAULT_LONGITUDE=144
    - OPENVPNMONITOR_DEFAULT_MAPS=True
    - OPENVPNMONITOR_DEFAULT_MAPSHEIGHT=500
    - OPENVPNMONITOR_DEFAULT_SITE=Sabo VPN
    - OPENVPNMONITOR_SITES_0_ALIAS=UDP
    - OPENVPNMONITOR_SITES_0_HOST=<LOCAL_IP>
    - OPENVPNMONITOR_SITES_0_NAME=UDP
    - OPENVPNMONITOR_SITES_0_PORT=<PORT>
    - OPENVPNMONITOR_SITES_0_SHOWDISCONNECT=True
```

Zde je tedy pro nasazení použit image *ruimarinho/openvpn-monitor* sice od jiného distributora než samotná instance OpenVPN, avšak je tento OVPN monitor přímo konstruovaný na napojení k OpenVPN. Dále je zde důležitých několik proměnných. Dost z těchto proměnných má spíše minoritní roli a neodvíjí se od nich přímo funkčnost samotného řešení jako například nastavení formátu data, defaultní nastavení zobrazení mapy atp. Ale je zde několik proměnných začínajících *OPENVPNMONITOR_SITES_0…*, které již jsou zásadní pro následné připojení k OpenVPN instanci. Zde se udává metoda připojení, v tomto případě UDP, dále port, na kterém je vystvený tzv. management OVPN a lokální IP adresa pro možné spojení s OpenVPN. Vzhledem k tomu, že samotnou instanci OpenVPN nebylo možné nasadit do Dockeru použitím souboru *docker-compose.yml*, nejsou tyto Dockercontainery po nasazení ve stejné lokální síti. Je zde tedy následně nutné OVPN monitor od jeho přidělené defaultní lokální síti odpojit a manuálně jej následně připojit k OpenVPN. Tím je zajištěna funkčnost UI viz.:




*Obrázek 22 – OVPN monitor*

Na *Obrázku 22* je tedy vidět samotné funkční řešení VPN monitoringu. Zde jsou vidět na prvním řádku obecné informace o stavu hostu dané OpenVPN. Pod tímto řádkem se již nachází tabulka přehledu samotných aktivních VPN certifikátů. Adresy a názvy zde musí být cenzurovány vyjma názvu certifikátu autora této práce. V tomto tabulkovém přehledu se tedy nachází názvy certifikátů, jejich IP adresy, network traffic směrem dovnitř i ven, datum kdy byl certifikát naposledy dostupný, čas po kterou dobu je již certifikát online nebo třeba také lokace nasazení daného certifikátu, vyobrazené také na mapě pod touto tabulkou.

# 5 Výsledky a diskuse

V této závěrečné kapitole bude shrnuto a zhodnoceno implementované řešení. Je nutno dodat, že toto vytvořené řešení monitoringu IT prostředků je již jistou dobu nasazeno a je možné dojít k určitým závěrům. Doba, po kterou je toto řešení nasazeno, se však nedá přesně uvést. Tato doba může být i v rámci implementovaných řešení poměrně odlišná. Některé nástroje a jejich řešení jsou již nasazeny delší dobu, jiná kratší. Doba, kdy byla jednotlivá řešení nasazována je v rozsahu uplynulých dvou let. U některých nástrojů nebo řešení byly například vyžadované dodatečné úpravy pro jejich zefektivnění nebo odstranění problémů.

## 5.1 Ověření funkčnosti a spolehlivosti implementovaných řešení

Funkčnost implementovaného řešení byla ověřena především faktorem času, kdy některá řešení byla již zpočátku funkční, jiná vyžadovala následnou opravu po zjištění určitých chyb a nedostatků.

V souhrnu je možné označit za nejspolehlivější část implementace tu, která využívá architekturu s nástrojem Prometheus. Spolehlivost je dána především předdefinovanými metrikami v rámci jednotlivých exporterů, dále jednoduchou a bezproblémovou konfigurací nástroje Prometheus a následně většinou využívanými předdefinovanými dashboardy v Grafaně. Tyto předdefinované dashboardy, dostupné z platformy Grafana Labs, jsou již prověřeny delším časem a několika tisíci uživateli s mírnou customizací. Zde však ve většině případů samotná monitorovací řešení fungovala již od prvotního nasazení bez nutnosti jejich úprav. Je nutné pouze pravidelně aktualizovat konfigurační soubor nástroje Prometheus pro aktualizaci sledovaných serverů a containerů dle potřeb. Co zde však byl problém, který bylo následně třeba ošetřit, je container-exporter nasazený v Dockeru. Ten měl defaultně špatně vyřešené cachování dat a postupem času zde byl zaznamenán (právě díky monitoringu) konstantní nárůst využití RAM. Z tohoto důvodu bylo nutné zavést restrikci využití RAM tohoto containeru na 1G. Další problémy byly stejně tak s alertingem. To se však netýká pouze této architektury, ale i všech ostatních, kde byl alerting nasazen. Jednalo se například o nevhodné nastavení času, po který musí být hodnota v tresholdu, aby Grafana odeslala notifikaci. Především v případě, kdy byl tento časový interval příliš krátký, byly zasílány notifikace ukazující error, které však nebyly validní. K tomu mohlo dojít pouze náhlým

výkyvem do tresholdu, což byl dostačující spouštěč pro nevalidní alert. Tento problém byl vyřešen prodloužením časového intervalu, po který musely být hodnoty v předdefinovaném tresholdu.

Další architektura za použití nástroje InfluxDB již byla trochu poruchovější, avšak stále vcelku spolehlivá s maximálně pár drobnými úpravami po nasazení a následném ověřování spolehlivosti faktorem času. Zde se objevovaly určité problémy především z důvodů vytváření vlastních scriptů pro sběr dat. Tyto scripty občas vyžadovaly sledování a ověřování po nějakou dobu, jestli fungují tak jak mají. V případě zjištění těchto nedostatků bylo nutné scripty následně doladit. Nejproblematičtější zde bylo řešení týkající se monitoringu SSL certifikátů, které následně vyžadovalo několik úprav. Toto řešení také vyžadovalo aktualizaci počtu certifikátů zasílaných do nástroje InfluxDB, neboť zde musí být uveden přesný počet řádků dle počtu sledovaných DNS a není možné zasílat prázdné hodnoty. Krom toho však tato architektura funguje vcelku bezchybně.

Další, vzhledem ke spolehlivosti již náročnější, použitá architektura je za použití nástroje Loki. U tohoto řešení nebyla takovým problém samotná konfigurace nebo funkčnost nástrojů, avšak obrovské množství získávaných dat v rámci aplikačních logů. S obrovským objemem získávaných dat nástroj neustále zatěžoval využití diskového prostoru na hostovaném serveru. Vzhledem k tomu musel být nástroj Loki, pro zamezení výpadků ostatních aplikací a nástrojů na daném serveru, přesunut na oddělený server určený pouze pro tento nástroj. Navzdory tomu byl však nárůst využití disku stále konstantně rostoucí. Pro konečné dořešení tohoto problému byly vytvořeny příkazy ve službě Cron, které každý den mazaly data starší 14 dní. Následně se zde objevoval již pouze problém týkající se samotného loadování získaných dat v Grafaně. Běžně se jedná o několik log lines za minutu. Kvůli tomu je vhodné v Grafaně zobrazovat maximálně hodinové časové úseky. Při delším časovém intervalu se již může stát, že Grafana skončí loadování s timeout errorem, neboť toto obrovské množství dat nestihne načíst. Kromě toho však dané řešení funguje a splňuje zadané požadavky.

Dále nástroj Portainer i s jeho agenty dodnes nevykazoval žádné chyby nebo nedostatky. Toto řešení bezchybně funguje již od počátku nasazení. Avšak se nedá vyloženě mluvit o nějaké komplexní architektuře, neboť toto řešení se skládá pouze z jednoho nástroje a k němu napojených agentů přímo vytvořených pro kompatibilitu s nástrojem Portainer.

U posledního řešení monitoringu aktivních VPN certifikátů se vyskytoval pouze jeden problém, který bylo po zjištění nefunkčnosti monitoringu potřeba řešit. Problém spočíval v tom, že pro funkčnost řešení je nezbytně vyžadováno, aby Docker containery samotné VPN a jeho monitoringu byly ve stejné lokální síti. Jelikož samotná VPN však nebyla nasazována pomocí docker-compose, bylo nutné toto spojení zajistit manuálně. Toto řešení následně funguje, dokud například některý z containerů nespadne. Containery jsou sice nastaveny tak, aby se samy ihned restartovaly a koncový uživatel většinou ani nepozná, že k nějakému výpadku došlo. Avšak po restartu se tato lokální síť přeruší a je třeba containery manuálně spojit v lokální síti znovu. Toto byl však jediný problém daného řešení. Kromě toho stejně jako všechny předešlé implementace dále funguje v produkčním prostředí a velice usnadňuje interní procesy firmy.

## 5.2 Hodnoty monitoringu

Hodnoty implementovaných řešení jsou vyobrazované různě. U veliké části panelů jsou například pro okamžitý přehled pro všechny zůčastněné subjekty hodnoty převedeny na %. Na této stupnici od 0 do 100 jsou standardně nastavovány tresholdy pro alerting v rozmezí 80 – 95 % pro včasné povšimnutí a možnost odstranění problémů před dosažením jejich kritické hranice. U některých panelů však hodnoty zůstávají vyobrazené v jednotkách určených k dané metrice. V této práci je touto jednotkou ve většině případů byte v jeho různých variacích, jako například mega, giga, za sekundu apod. Ne všechna řešení a panely však vykazují takovéto hodnoty. Další možný výskyt hodnot je typu boolean, většinou uváděný 1 / 0. To je běžně přepisováno v UI dashboardu na UP / DOWN. Mimo to se zde vyskytují také nečíselné hodnoty, jako například loglines, u kterých můžou být v rámci zobrazování vyhledávána jejich klíčová slova jako INFO, DEBUG, ERROR apod.

### 5.2.1 Využití naměřených hodnot

Tyto získané hodnoty pak v praxi slouží především pro vlastní zaměstnance podniku, ale i další subjekty, kterými mohou být klienti. Prvním významným využitím je zde již několikrát zmiňované vytváření alertingu, které vede ke včasnému odhalení blížících se incidentů a možnosti jejich předcházení. Dále tyto naměřené údaje slouží pro možnost analyzování systémů a celé infrastruktury. Na základě toho je možné například efektivně přidělovat zdroje určitým aplikacím, plánovat využití zdrojů, odhadovat robustnost systémů

apod. Dále však také velice dobře slouží konkrétním vývojářům daných aplikací, kteří mají díky monitoringu možnost bezproblémového přístupu k vnitřní struktuře jejich nasazených aplikací. Takovými možnostmi může být například již zmiňované debuggování chyb na základě aplikačních logů, možnost sledovat chování jejich aplikací vzhledem ke spotřebě zdrojů, odhalovat potenciální problémy a zároveň jim předcházet apod. Dalším koncovým uživatelem vytvořeného monitoringu může být také klient, který má možnost a přístup k efektivnímu přehledu chování pro něj implementovaného produktu. Tento přehled může být například formou generování měsíčních reportů z Grafany a následné zasílání klientovi s přehledem chování aplikace za posledních 30 dnů. Nebo také formou přímého přístupu do nasazené Grafany s omezenými právy pouze pro možnost sledování jemu určenému dashboardu.

## 5.3 Ekonomické zhodnocení

Po ekonomické stránce má monitoring velký význam z hlediska šetření nákladů. V konečném důsledku se jedná o primární cíl v rámci toho, co by měl monitoring přinést. Vzhledem k požadavkům firmy však není možné uvádět konkrétní částky, o které se i vzhledem k šetření nákladů za využití monitoringu IT prostředků jedná. Vzhledem ke komplexnosti firemní infrastruktury, navíc i bez nutnosti cenzury, není možné říct ve všech případech zcela přesně, o jakou konkrétní částku vzhledem k šetření nákladů se jedná. Avšak je možné uvádět přibližná procenta, o které monitoring snížil firemní náklady.

Nejzásadnějším faktorem, vzhledem k šetření nákladů, zvýšení zisků a celkově zefektivnění firemních procesů, je fakt, že většinu zaměstnanců firmy (především vývojářů) je možné alokovat na jiné, produktivnější a výdělečnější úkoly. To monitoring zajišťuje zejména tím, že poskytuje právě vývojářům efektivní, přehledné, a hlavně přístupné prostředí pro jednoduché debuggování jejich aplikací. Díky tomu mají tyto úkony podstatně rychleji za sebou a mohou pokračovat s dalším vývojem. Dále také vývojáři již nemusejí kontrolovat své aplikace, jestli fungují jak mají. Nebo jestli třeba nehrozí díky nevyladěnému využití zdrojů k zahlcení systému a tím nedochází k pádu jejich aplikací na daném hostu. O toto se stará klasický monitoring prostředků, zefektivněný jeho alertingem. Vzhledem k možnosti alokování vývojářů, jejichž hodinová sazba bývá zpravidla vysoká, na produkční vývoj, místo řešení těchto činností tak daný proces zlepšuje ekonomickou situaci firmy. Samozřejmě zaměstnanci jsou placeni stále stejně, avšak za práci, která přináší do firmy

větší zisky. V tomto případě je tedy možné mluvit spíše o stálých nákladech, avšak o zvýšení zisků. Vzhledem k času, který by vývojáři strávili na manuálních a repetitivních úkolech, který nyní zajišťuje monitoring, můžeme v konečném důsledku mluvit o 10 – 15% navýšení zisků.

Další významný faktor, který se v konečném důsledku projeví na zlepšení ekonomické situace podniku, je mnohem vyšší možnost zamezení výpadků systému a jejich aplikací. To je zajištěno především včasným alertingem, který umožňuje provést úpravy včas před zaplněním zdrojů systému, které výpadku zabrání. Zde může být přínos do ekonomického zhodnocení velice odlišný v závislosti na různorodosti prostředí serverů. Každý server totiž hostuje různý počet aplikací a služeb s různými funkcemi. Je pak tedy velice odlišné z hlediska ekonomiky, jestli by se mělo jednat o pád serveru s několika produkčními aplikacemi, které klient aktivně používá, nebo o server s primárně nasazenými pouze pomocnými službami. V případě produkčního serveru se pak také jedná o čas, kdy by k výpadku mohlo dojít. Je veliký rozdíl pokud aktuálně používaná aplikace spadne během vytíženého času v pracovní dny nebo o víkendu v noci. Pokud by však byl brán v potaz server právě s takovýmito produkčními aplikacemi a mělo by dojít k pádu bez předcházejícího povšimnutí v době vysoké vytíženosti, mohou být následky včetně ekonomického faktoru fatální. Může dojít dokonce i k přerušení právě probíhajících procesů na straně klienta, například v návaznosti na databázi bez možnosti opakování. V tomto případě mohou dle smluvních pravidel následovat v nejhorším případě dokonce veliké finanční sankce. Takovéto sankce mohou být různé, avšak v horším případě by bylo možné počítat s nárůstem nákladů až o 20 %.

Další úspory nákladů zajišťuje možnost analýzy stávajících systémů a využití jejich prostředků. Na základě této analýzy je možné nahlédnout do přehledu všech využívaných serverů s vyobrazením jejich stavu využití IT zdrojů. Z toho je možné zpracovat analýzu, která může vést k efektivnímu přerozdělení aplikací a služeb podle využití zdrojů, případně přidávání nových služeb na takové servery, které mají stále dostatek volných prostředků. To zamezí nutnosti placení zbytečně velikého množství VPS, případně spravování zbytečně velké infrastruktury fyzických serverů, které by s sebou nesly zvýšení nákladů. Pokud by nebyl tento monitoring zřízen, platila by firma sice v jednotkách, ale přesto větší počet spravovaných serverů. Ty by neměly efektivně využité zdroje a spoustu z nich by nebyly

naplno využité. S přihlédnutím k šetření nákladů, díky možné eliminaci placení zbytečného množství serverů navíc, je zde možné mluvit o zhruba 3% šetření nákladů.

Dále je možné v rámci ekonomického zhodnocení mluvit i o částech monitoringu, které interně pro firmu Sabo nemají konkrétní využití, avšak jsou smluvně vyžadovány klientem. V takovém případě je možné mluvit o navýšení zisků. Navýšení může být zajištěno buďto celkovou částkou za projekt, kdy klient může monitoring bezpodmínečně vyžadovat v rámci výběru dodavatele pro vyvíjený projekt, nebo jako příplatek klienta za dodatečné zpracování požadovaného monitoringu. Těmito částmi monitoringu mohou být například pravidelné měsíční reporty nebo jiná různě specifická řešení. Zde je opět velice obtížné odhadovat konkrétní nárůst zisků, v závislosti od požadavků nebo podmínek klienta a velikosti samotného projektu. V menších případech u dodatečného požadavku na zpracování monitoringu se může jednat o 2% navýšení zisků. Avšak s přihlédnutím k faktu, že monitoring může být klientovou podmínkou při výběru dodavatele zakázky, je možné mluvit i o celé částce za projekt, což v případě velikého projektu může být až 25 % celkových zisků firmy.

# 6 Závěr

V této diplomové práci, která se zabývá problematikou monitoringu IT prostředků, bylo úspěšně dosaženo všech stanovených cílů. Nejprve byla úvodem této práce zpracována teoretická část, která vedla především díky obsažené analýze dostupných monitorovacích nástrojů ke stanovení základních poznatků a vstupů pro následující vlastní práci.

V této analýze bylo vybráno několik monitorovacích nástrojů, především z řad těch známějších, a tyto nástroje byly následně analyzovány. U každého byly uvedeny jeho jednoznačné výhody či nevýhody, především ve vztahu k této práci a požadavkům firmy SABO Mobile IT s.r.o. Na základě těchto uvedených aspektů byl každý z nástrojů doporučen pro následnou implementaci ve vlastní práci, či naopak zavrhnut.

Ve vlastní práci byl nejprve vybrán a představen zmíněný podnik Sabo. Mimo obecné představení byly uvedeny také prostředky ve vztahu k této práci, se kterými firma pracuje. Jedná se především o Linuxové servery a další části podnikové infrastruktury. Následně byl analyzován již využívaný monitoring firmou Sabo. Na základě získaných informací bylo možné následně stanovit návrh monitorovacích řešení pro následnou implementaci dle use cases vycházejících z potřeb podniku.

Dále byly na základě stanovených řešení a cílů navrženy konkrétní architektury, které by bylo vhodné implementovat. Konkrétně se jedná prvotně o nasazení nové instance Grafany, která by měla sloužit jako primární vizualizační nástroj. Dále v rámci architektur, které zajišťují sběr dat, je první navrženou za využití nástroje Prometheus pro uchovávaní a distribuci získaných dat s využitím různých druhů exporterů pro samotný sběr dat ze systémů. Další navržená architektura je za využití nástroje InfluxDB pro uchovávání a distribuci dat typu časové řady s vytvářením vlastních scriptů v prostředí bash pro sběr dat dle specifických use cases. Další architektura, která byla navržena využívá nástroje Loki pro uchovávání a distribuci aplikačních logů za využití nástroje Promtail pro jejich možný sběr. Všechny tyto zmíněné architektury budou distribuovat data do Grafany pro sjednocení vizualizací do jedné platformy. Dále byly navrženy také architektury, které neumožňují distribuci dat do Grafany, avšak mají vlastní efektivní vizualizační metody. První z těchto navržených architektur je Portainer, využívající vlastní UI pro možný monitoring a efektivní správu Docker containerů, za použití jeho agentů pro sběr dat. Druhou architekturou s vlastním vizualizačním UI je monitoring VPN certifikátů služby OVPN.

Všechny tyto zmíněné architektury byly následně implementovány s pokrytím jednoho nebo více stanovených use cases. Implementace byla provedena formou nasazení zmiňovaných nástrojů a služeb do produkčního prostředí firmy Sabo, především skrze využití VPS Linuxového serveru, který byl vytvořen pro tuto práci. Nástroje byly kromě vestavěných služeb v systému Linux v drtivé většině nasazeny jako Docker containery především pro usnadnění konfigurací, lepší škálovatelnost a zefektivnění využití zdrojů.

Toto implementované řešení bylo závěrem ověřeno a zhodnoceno. Jako první byla ověřena funkčnost a spolehlivost nasazených řešení včetně popisu nutných dodatečných úprav, které byly ve většině případů dostačující pro zajištění spolehlivého, a především automatizovaného chodu. Byly také uvedeny a vysvětleny získávané hodnoty odlišných implementovaných řešení. Ke každému z typů těchto hodnot byla také uvedena jejich konkrétní možná využití v praxi. Na základě ověřování efektivity faktorem času, zkušenostmi a v rámci možností získanými daty od vedení firmy bylo úplným závěrem stanoveno ekonomické zhodnocení. To bylo pojato z několika možných pohledů, neboť funkce automatizovaného monitoringu umožňuje více než jedno využití. Ekonomické zhodnocení bylo provedeno, vzhledem k nutné cenzuře konkrétních částek a často nemožné zjištění přesných částek, které byly přímo ovlivněny monitoringem, formou přibližného procentuálního nárůstu zisků nebo snížení nákladů. Konkrétními faktory zhodnocení pak je možná alokace vývojářů na produktivnější činnosti, předcházení výpadkům nasazených aplikací a služeb, možnost analýzy systémů pro efektivnější alokaci nasazovaných aplikací vzhledem k využití zdrojů a také splnění smluvních závazků a požadavků klientů na zřízení různých forem monitoringu.

# 7 Seznam použitých zdrojů


1) TURNBULL, James, 2016. The Art of Monitoring. Paperback. ISBN 9780988820241.

2) JANÍK, David, 2021. Jak na měření využití CPU v Linuxu? Váš hosting [online]. Dostupné z: https://www.vas-hosting.cz/blog-jak-na-mereni-vyuziti-cpu-v-linuxu

3) Co je systémová paměť (RAM), 2023. DELL Zechnologies [online]. 6. Dostupné z: https://www.dell.com/support/kbdoc/cs-cz/000148441/co-je-systémová-paměť-ram

4) ZIVANOV, Sara, 2023. Swap Memory: What It Is, How It Works, and How to Manage It. Phoenix NAP [online]. Dostupné z: https://phoenixnap.com/kb/swap-memory

5) Disk performance metrics, 2024. Microsofi Learn [online]. Dostupné z: https://learn.microsoft.com/en-us/azure/virtual-machines/disks-metrics

6) WILSON, Jeff, 2024. What is Uptime? Overview, Importance, Tips and Best Practices. RoseHosting [online]. Dostupné z: https://www.rosehosting.com/blog/what-is-uptime-overview-importance-tips-and-best-practices/

7) ENGLISH, Melanie, 2022. 20 Ways to Monitor Network Traffic: Steps, Tips and Tools. TERAMIND [online]. Dostupné z: https://www.teramind.co/blog/ways-to-monitor-network-traffic/

8) LAMBERTI, Alyssa, 2023. 19 Network Metrics: How to Measure Network Performance. Obkio [online]. Dostupné z: https://obkio.com/blog/how-to-measure-network-performance-metrics/

9) BEN-ARI, Demi, 2023. What is Cyber Security Monitoring? Panorays [online]. Dostupné z: https://panorays.com/blog/what-is-cyber-security-monitoring/

10) STOLTZFUS, Justin, 2022. What's the difference between SEM, SIM and SIEM? Techopedia [online]. Dostupné z: https://www.techopedia.com/7/31201/security/whats-the-difference-between-sem-sim-and-siem

11) What Is Security Monitoring? Importance and Tools. Sapphire [online]. Dostupné z: https://www.techopedia.com/7/31201/security/whats-the-difference-between-sem-sim-and-siem

12) What Is Security Monitoring? Importance and Tools. LogicMonitor [online]. Dostupné z: https://www.logicmonitor.com/blog/what-is-database-monitoring-and-why-is-it-still-important

13) What Is Application Performance Monitoring? Cisco [online]. Dostupné z: https://www.appdynamics.com/topics/what-is-APM

14) KAMMER, Michael, 2023. What is Server Monitoring? Everything You Need to Know. Zabbix Blog [online]. Dostupné z: https://blog.zabbix.com/what-is-server-monitoring-everything-you-need-to-know/26617/

15) PLESKY, Elvis, 2021. What is Server Monitoring? Plesk [online]. Dostupné z: https://www.plesk.com/blog/various/server-monitoring-overview/

16) MANI, Ganesh, 2021. Container Monitoring: Essential Tools + Best Practices. Scout APM [online]. Dostupné z: https://scoutapm.com/blog/container-monitoring

17) SABHARWAL, Navin a Piyush PANDEY, 2020. Monitoring Microservices and Containerized Applications. 1. Apress L. P. ISBN 9781484262160.

18) Container Monitoring. AWS [online]. Dostupné z: https://aws.amazon.com/cloudwatch/container-monitoring/

19) SPASOJEVIC, Anastazija, 2023. What is Cloud Monitoring? Definition and Best Practices. Phoenix NAP [online]. Dostupné z: https://phoenixnap.com/blog/cloud-monitoring

20) SHARIF, Arfan, 2023. WHAT IS CLOUD MONITORING? Crowdstrike [online]. Dostupné z: https://www.crowdstrike.com/cybersecurity-101/observability/cloud-monitoring/

21) ANGELAS, 2024. What is Cloud Monitoring? How to Make Sure Cloud Services are Working Properly. Stackify [online]. Dostupné z: https://stackify.com/cloud-monitoring/

22) WICKRAMASINGHE, Shanika, 2023. Active vs. Passive Monitoring: What's The Difference? Splunk [online]. Dostupné z: https://www.splunk.com/en_us/blog/learn/active-vs-passive-monitoring.html

23) Active vs. Passive Monitoring: what's the difference & why it matters, 2022. Red Sift [online]. Dostupné z: https://blog.redsift.com/brand-protection/active-vs-passive-monitoring-whats-the-difference-why-it-matters/

24) LAFLAMME, Ryan. Agent vs Agentless Monitoring: Which is Best? Auvik [online]. Dostupné z: https://www.auvik.com/franklyit/blog/agent-vs-agentless-monitoring/

25) Agent Vs Agentless Monitoring: What's The Difference?, 2021. ZIF [online]. Dostupné z: https://zif.ai/agent-vs-agentless-monitoring-whats-the-difference/

26) CHATTERJEE, Shormistha, 2023. What is End-to-End Monitoring? BrowserStack [online]. Dostupné z: https://www.browserstack.com/guide/end-to-end-monitoring

27) DEMPSEY, Kelley, Nirali CHAWLA, L. JOHNSON, Ronald JOHNSTON, Alicia JONES, Angela OREBAUGH, Matthew SCHOLL a Kevin STINE, 2011. Information Security Continuous Monitoring (ISCM) for Federal Information

Systems and Organizations. NIST SP 800-137. NIST [online]. Dostupné z: https://csrc.nist.gov/pubs/sp/800/137/final

28) CYBERSECURITY. NIST [online]. Dostupné z: https://www.nist.gov/cybersecurity

29) ISO/IEC 27001:2022, Information security, cybersecurity and privacy protection. ISO [online]. Dostupné z: https://www.iso.org/standard/27001

30) ITIL® 4: the framework for the management of IT-enabled services. Axelos [online]. Dostupné z: https://www.axelos.com/certifications/itil-service-management

31) Data protection in the EU, 2018. European Commission [online]. Dostupné z: https://commission.europa.eu/law/law-topic/data-protection/data-protection-eu_en

32) DOUPAL, František, 2023. Význam automatizace a generativní AI ve firemním IT roste. SALESFORCE. Reseller Magazine OnLine [online]. Dostupné z: https://www.rmol.cz/novinky/vyznam-automatizace-generativni-ai-ve-firemnim-it-roste

33) Inteligentní automatizace jako cesta k úspěšné firmě budoucnosti. Deloitte [online]. Dostupné z: https://www2.deloitte.com/cz/cs/pages/strategy-operations/articles/inteligentni-automatizace-jako-cesta-k-uspesne-firme-budoucnosti.html

34) Monitoring Automation. Hewlett Packard Enterprise [online]. Dostupné z: https://docs.microfocus.com/OMi/10.62/Content/OMi/ConceptsGuide/getStarted/getStarted_concepts_MA.htm

35) BOHUSLAV, Daniel, 2019. DevOps – opravdu dělá práci efektivnější? Memos [online]. Dostupné z: https://www.memos.cz/devops-opravdu-dela-praci-efektivnejsi/

36) GONZALEZ, Alberto, 2023. Linux Server Cookbook. 1. BPB Publications. ISBN 9789355513588.

37) Understanding Linux, 2023. Red Hat [online]. Dostupné z: https://www.redhat.com/en/topics/linux

38) KIARIE, James, 2023. What Is Linux? and How Does Linux Work? Tec Mint [online]. Dostupné z: https://www.tecmint.com/what-is-linux/

39) Linux – proč je lepší?, 2016. Linux Mint [online]. Dostupné z: https://www.linux-mint-czech.cz/2016/01/linux-proc-je-lepsi/

40) VON HARZ, Tyler, 2016. The Complete History of Linux: Everything You Need to Know. History computer [online]. Dostupné z: https://history-computer.com/the-complete-history-of-linux-everything-you-need-to-know/

41) LACROIX, Jay, 2022. Mastering Ubuntu Server. 4. Packt Publishing, Limited. ISBN 9781803235042.

42) The Ubuntu lifecycle and release cadence. Ubuntu [online]. Dostupné z: https://ubuntu.com/about/release-cycle

43) VELIMIROVIC, Andreja, 2023. Debian vs. Ubuntu Server. Phoenix NAP [online]. Dostupné z: https://phoenixnap.com/blog/debian-vs-ubuntu-server

44) Ubuntu vs. Debian: Which one is better?, 2023. IONOS [online]. Dostupné z: https://www.ionos.com/digitalguide/server/know-how/ubuntu-vs-debian/

45) Virtuální vs. fyzický server, 2023. LevnaPC [online]. Dostupné z: https://www.levnapc.cz/virtualni-vs-fyzicky-server.html

46) Jaké jsou hlavní výhody a nevýhody použití VPS oproti dedikovanému serveru? MyDreams [online]. Dostupné z: https://www.mydreams.cz/cz/wiki/181-jake-jsou-hlavni-vyhody-a-nevyhody-pouziti-vps-oproti-dedikovanemu-serveru.html

47) JANÍK, David, 2017. Výhody dedikovaného serveru (VDS) oproti VPS. Váš hosting [online]. Dostupné z: https://www.vas-hosting.cz/blog-vyhody-dedikovaneho-serveru-vds-oproti-vps

48) RYDER, Tom, 2018. Bash Quick Start Guide. 1. Packt Publishing, Limited. ISBN 9781789534085.

49) CAMPESATO, Oswald, 2022. Bash for Data Scientists. 1. Mercury Learning & Information. ISBN 9781683929727.

50) KOFLER, Michael, 2024. Scripting. 1. Rheinwerk Publishing. ISBN 9781493225576.

51) SCHKN. Cron Jobs and Crontab on Linux Explained. Dev content [online]. Dostupné z: https://devconnected.com/cron-jobs-and-crontab-on-linux-explained/

52) BORISOV, Bobby, 2024. How to Use Cron on Linux: Tips, Tricks, and Examples. Linuxiac [online]. Dostupné z: https://linuxiac.com/how-to-use-cron-and-crontab-on-linux/#What_is_Cron

53) What Is NGINX? Nginx [online]. Dostupné z: https://www.nginx.com/resources/glossary/nginx/

54) FJORDVALD, Martin Bjerretoft a Clement NEDELCU, 2018. Nginx HTTP Server. 4. Packt Publishing, Limited. ISBN 9781788621977.

55) KABELOVA, Alena a Libor DOSTALEK, 2006. DNS in Action. 1. Packt Publishing, Limited. ISBN 9781847190635.

56) OPPLIGER, Rolf, 2009. SSL and TLS. 1. Artech House. ISBN 9781596934481.

57) HICKS, Richard M., 2021. Implementing Always on VPN. 1. Apress L. P. ISBN 9781484277416.

58) MULI, Joseph, 2018. Beginning DevOps with Docker. 1. Packt Publishing, Limited. ISBN 9781789539578.

59) SHARMA, Hitesh Kumar, Anuj KUMAR, Sangeeta PANT a Mangey RAM, 2023. DevOps. 1. River Publishers. ISBN 9781003807728.

60) GHOSH, Saibal, 2020. Docker Demystified. 1. BPB Publications. ISBN 9789389845884.

61) POULTON, Nigel, 2023. Docker Deep Dive. 2. Packt Publishing, Limited. ISBN 9781835087916.

62) HERNANDEZ, Rodrigo Juan, 2022. Building IoT Visualizations Using Grafana. 1. Packt Publishing, Limited. ISBN 9781803231938.

63) SALITURO, Eric, 2023. Learn Grafana 10. x. 2. Packt Publishing, Limited. ISBN 9781801814331.

64) MCCOLLAM, Ronald, 2022. Getting Started with Grafana. 1. Apress L. P. ISBN 9781484283097.

65) CHAPMAN, Rob a Peter HOLMES, 2024. Observability with Grafana. 1. Packt Publishing, Limited. ISBN 9781803249643.

66) BASTOS, Joel a Pedro ARAÚJO, 2019. Hands-On Infrastructure Monitoring with Prometheus. 1. Packt Publishing, Limited. ISBN 9781789808032.

67) SABHARWAL, Navin a Piyush PANDEY, 2020. Monitoring Microservices and Containerized Applications. 1. Apress L. P. ISBN 9781484262160.

68) What is Prometheus? Prometheus [online]. Dostupné z: https://prometheus.io/docs/introduction/overview/

69) EXPORTERS AND INTEGRATIONS. Prometheus [online]. Dostupné z: https://prometheus.io/docs/instrumenting/exporters/

70) InfluxDB Clustered. Influx data [online]. Dostupné z: https://www.influxdata.com/products/influxdb-clustered/

71) CICHY, Jakub, 2022. Hands-on InfluxDB. Software MILL [online]. Dostupné z: https://softwaremill.com/hands-on-influxdb/

72) Grafana Loki documentation. Grafana [online]. Dostupné z: https://grafana.com/docs/loki/latest/

73) Loki overview. Grafana [online]. Dostupné z: https://grafana.com/docs/loki/latest/get-started/overview/

74) PIECHNIK, Grzegorz, 2023. Loki: Effective Logging and Log Aggregation with Grafana. Medium [online]. Dostupné z: https://medium.com/@gpiechnik/loki-effective-logging-and-log-aggregation-with-grafana-c3356e7f13ad

75) LEVAN, Michael, 2022. Mythbusting Portainr - Setting the Record Straight. Portainer io [online]. Dostupné z: https://www.portainer.io/blog/mythbusting-portainer-so-much-more-than-a-docker-gui

76) Portainer architecture. Portainer Documentation [online]. Dostupné z: https://docs.portainer.io/start/architecture

77) HOROVITS, Dotan. The Complete Guide to the ELK Stack. Logz io [online]. Dostupné z: https://logz.io/learn/complete-guide-elk-stack/#latest-on-the-elk-stack

78) HAZEL, Thomas, 2023. 5 ELK Stack Pros and Cons. Chaos Search [online]. Dostupné z: https://www.chaossearch.io/blog/elk-stack-pros-and-cons

79) What is Sentry? Docs Sentry [online]. Dostupné z: https://docs.sentry.io/product/#what-is-sentry

80) SALA, Andrzej, 2023. Efficient error tracking with Sentry. Medium [online]. Dostupné z: https://medium.com/@AndrzejSala/efficient-error-tracking-with-sentry-e975c186947c

81) GILLIS, Alexander S., 2023. Nagios. TechTarget [online]. Dostupné z: https://www.techtarget.com/searchitoperations/definition/Nagios

82) WISNIOWSKI, Kamil, 2024. Nagios vs Zabbix – What's the Difference ? (Pros and Cons). Cloud Infractructure Services [online]. Dostupné z: https://cloudinfrastructureservices.co.uk/nagios-vs-zabbix-whats-the-difference/

83) MICHAŁ. How does the Zabbix monitoring system work? Blurify [online]. Dostupné z: https://blurify.com/blog/video-surveillance-the-control-and-safety-thanks-to-zabbix/

84) HATI, Spandita, 2023. What is Splunk? Tools, Architecture, Advantages. UpGrad [online]. Dostupné z: https://www.knowledgehut.com/blog/database/what-is-splunk

85) Introduction to Splunk, 2024. IntelliPaat [online]. Dostupné z: https://intellipaat.com/blog/what-is-splunk/

## 7.1 Zdroje použitých předdefinovaných Dashboardů

86) **Název dashboardu**: Node Exporter for Prometheus Dashboard based on 11074
**ID pro import**: 15172
**Autor**: wl4g
**Poslední aktualizace**: 28.11.2021
**Počet stažení**: 655 074
**Dostupné z**: https://grafana.com/grafana/dashboards/15172-node-exporter-for-prometheus-dashboard-based-on-11074/

87) **Název dashboardu**: Logs / App
**ID pro import**: 13639
**Autor**: Sadlil
**Poslední aktualizace**: 24.1.2022
**Počet stažení**: 1 596 795
**Dostupné z**: https://grafana.com/grafana/dashboards/13639-logs-app/

# 8 Seznam obrázků, tabulek, grafů a zkratek

## 8.1 Seznam obrázků



## 8.2 Seznam tabulek

## 8.3 Seznam použitých zkratek

IT = informační technologie

ICT = Information and Communication Technologies

VPS = virtual private server

HW = hardware

SW = software

DevOps = Development Operations

UI = User Interface

UX = User Experience

CPU = Central Processing Unit

GPU = Graphics Processing Unit

RAM = Random Access Memory

AWS = Amazon Web Services

GCP = Google Cloud Platform

SSD = Solid State Drive

HDD = Hard Disk Drive

IOPS = Input-output Operations Per Second

RAS = Remote Access Service

DDoS = Distributed Denial of Service

CISCO = networkingová organizace

HIPAA = Health Insurance Portability and Accountability Act

PCI DSS = Payment Card Industry Data Security Standard

SIM = Security Information Management

SEM = Security Event Management

SIEM = Security Information and Event Management

VM = Virtual Machine

DAM = Database Activity Monitoring

SQL = Structured Query Language

NoSQL = Not Only SQL

APM = Application Performance Monitoring

BE = Back End

FE =Front End

QoS = Quality of Service

SNMP = Simple Network Management Protocol

WMI = World Manufacturer Identifier

ICMP = Internet Control Message Protocol

HTTP = HyperText Transfer Protocol

HTTPS = Hypertext Transfer Protocol Secure

JDBC = Java Database Connectivity

NIST = National Institute of Standards and Technology

ISO = International Organization for Standardization

IEC = International Electrotechnical Commission

ISMS = Information Security Management Systém

ITIL = Information Technology Infrastructure Library

COBIT = Control Objectives for Information and Related Technology

GDPR = General Data Protection Regulation

DNS = Domain Name System

SSL = Secure Sockets Layer

JSON = JavaScript Object Notation

CORS = Cross-Origin Resource Sharing

UDP = User Datagram Protocol

MB = Megabyte

GB = Gigabyte

UFW = Uncomplicated Firewall

IoT = Internet of Things

GNU – GPL = GNU's Not Unix – General Public Licence

LTS = Long Term Support

CNCF = Cloud Native Computing Foundation

TSDB = Time Series Databases

API = Application Programming Interface

SDK = Software Development Kit

# Přílohy

**Příloha A** – Konfigurační soubor *prometheus.yml*

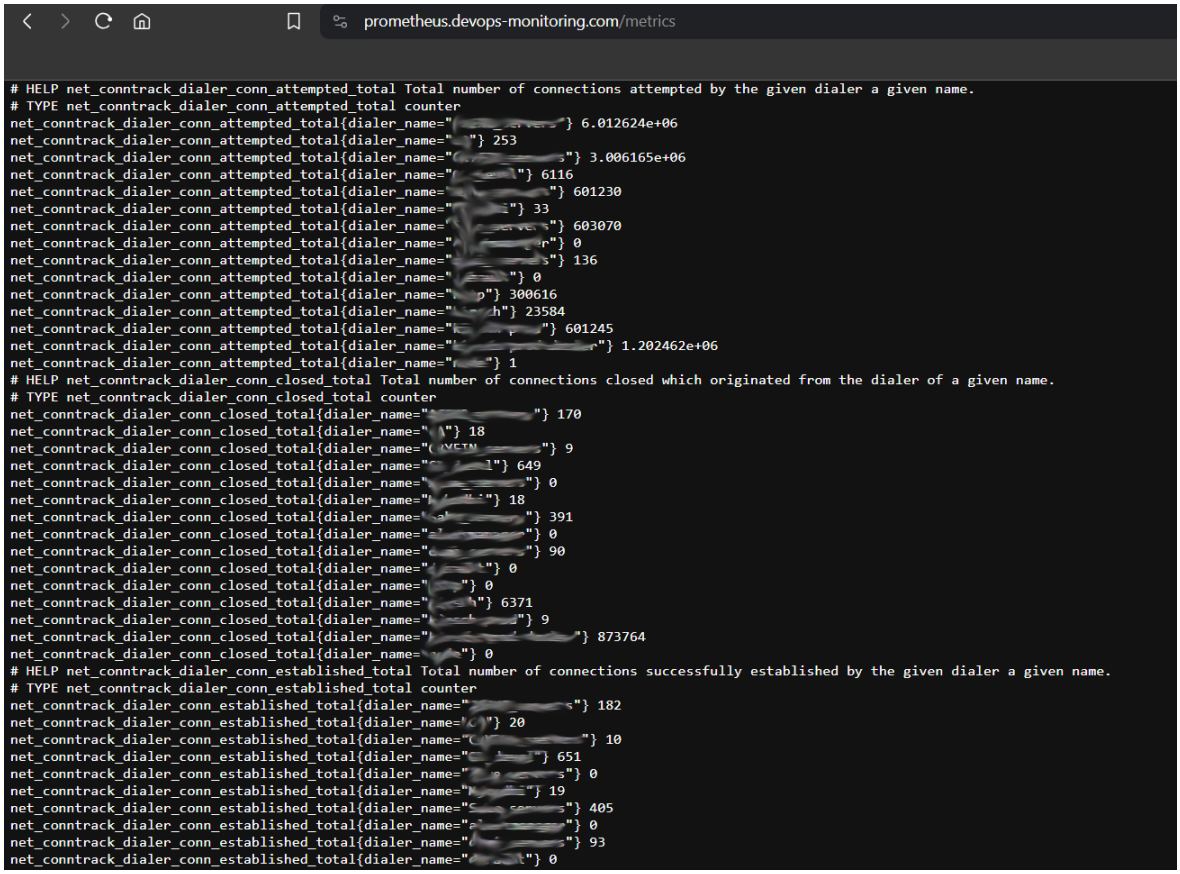
```
global:
  scrape_interval: 1m
  scrape_timeout: 10s
  evaluation_interval: 1m

alerting:
  alertmanagers:
  - follow_redirects: true
    enable_http2: true
    scheme: http
    timeout: 10s
    api_version: v2
    static_configs:
    - targets:
      - <IPv4>:<PORT>

rule_files:
- /etc/prometheus/alert.yml

scrape_configs:

- job_name: node
  honor_timestamps: true
  scrape_interval: 1m
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
    - node-exporter:<PORT>
    - container_exporter:<PORT>
    labels:
      instance: Devops_Monitoring

- job_name: Sabo_servers
  honor_timestamps: true
  scrape_interval: 30s
  scrape_timeout: 30s
  metrics_path: /metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  static_configs:
```

```
  - targets:
    - <IPv4>:<PORT>
    labels:
      instance: <HOST NAME>
  - targets:
    - <IPv4>:<PORT>
    labels:
      instance: <HOST NAME>
  - targets:
    - <IPv4>:<PORT>
    labels:
      instance: <HOST NAME>
#“a další targety”

- job_name: Miwe_servers
  honor_timestamps: true
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
    - <IPv4>:<PORT>
    labels:
      instance: <HOST NAME>
#“a další targety”

- job_name: ASEKO_servers
  honor_timestamps: true
  scrape_interval: 15s
  scrape_timeout: 10s
  metrics_path: /metrics
  scheme: http
  follow_redirects: true
  enable_http2: true
  static_configs:
  - targets:
    - <IPv4>:<PORT>
    labels:
      instance: <HOST NAME>
#“a další target + job namy”
```

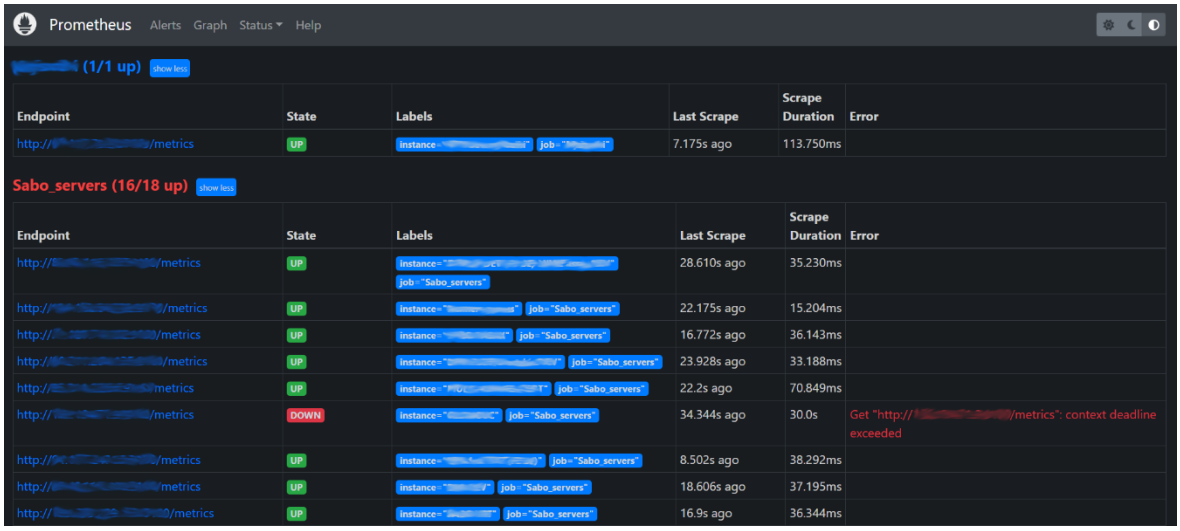
**Příloha B** – Konfigurační soubor Nginx pro nástroj Prometheus

```
server_name prometheus.devops-monitoring.com;

location / {
proxy_set_header Host $host;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-Host $host;
proxy_set_header X-Forwarded-Server $host;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
proxy_pass http://127.0.0.1:<PORT>;

auth_basic "prometheus";
auth_basic_user_file /etc/nginx/.htpasswd;
}
```

**Příloha C** – Consolové vyobrazení všech metrik nástrojem Prometheus

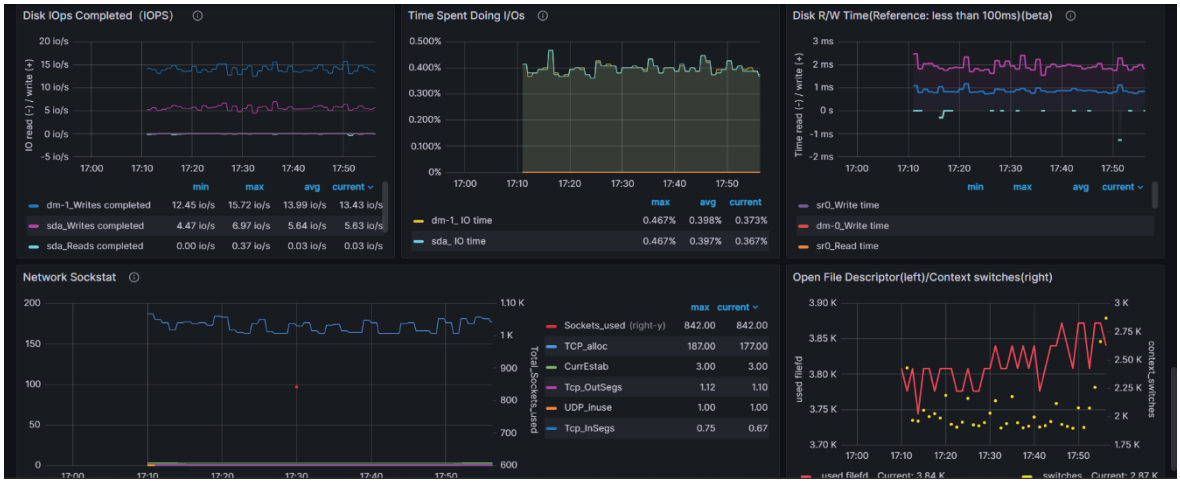


**Příloha D** – Vyobrazení sledovaných targets nástrojem Prometheus

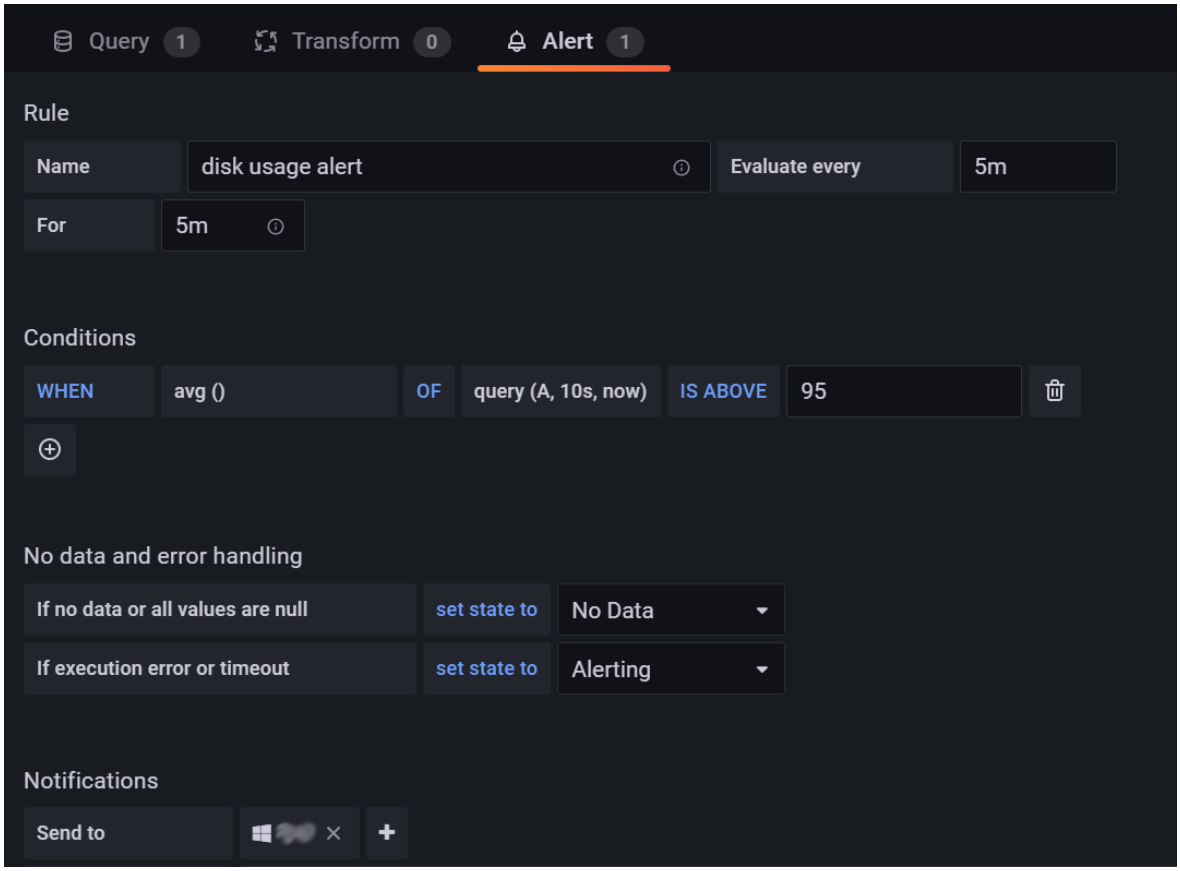



**Příloha E** – Dashboard přehledu všech serverů 3/4


**Příloha F** – Dashboard přehledu všech serverů 4/4



**Příloha G** – Nastavení alertů u dashboardu přehledu využití disku



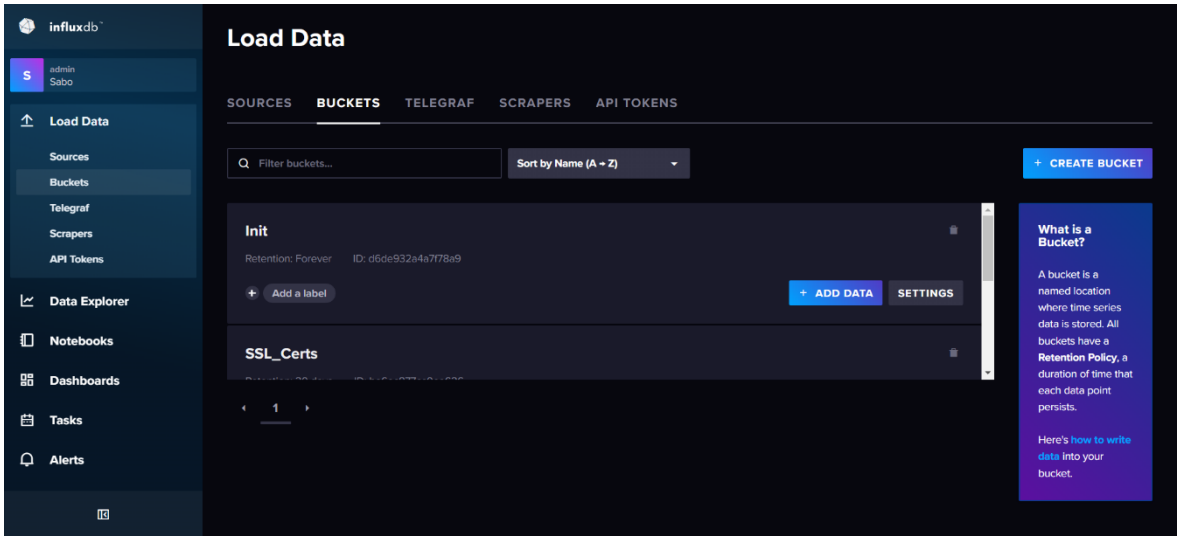

**Příloha H** – Konfigurační soubor Nginx pro nástroj InfluxDB

```
server {

        server_name influx.devops-monitoring.com;

        location / {

        proxy_pass          http://127.0.0.1:<PORT>;
        }
```

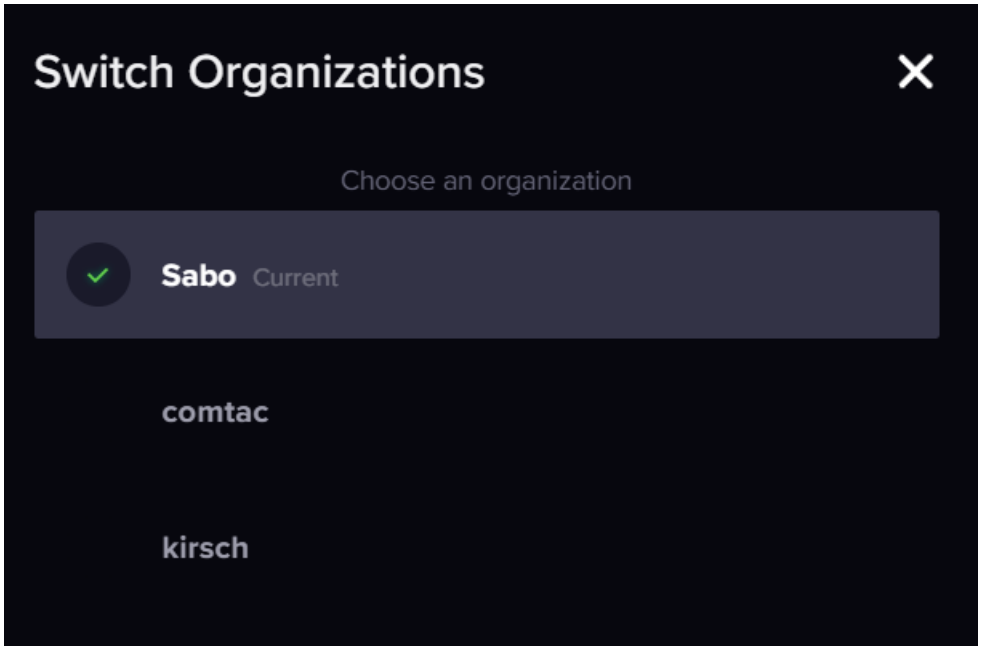
```
    listen [::]:443 ssl; # managed by Certbot
    listen 443 ssl; # managed by Certbot
    ssl_certificate /etc/letsencrypt/live/influx.devops-
monitoring.com/fullchain.pem; # managed by Certbot
    ssl_certificate_key /etc/letsencrypt/live/influx.devops-
monitoring.com/privkey.pem; # managed by Certbot
    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
```

```
}
server {
    if ($host = influx.devops-monitoring.com) {
        return 301 https://$host$request_uri;
    } # managed by Certbot



        server_name influx.devops-monitoring.com;

        listen 80;
        listen [::]:80;
    return 404; # managed by Certbot


}
```

**Příloha I** – Vyubrazení přehledu bucketů v nástroji InfluxDB



**Příloha J** – Přehled vytvořených organizací v nástroji InfluxDB




**Příloha K** – Bash script pro získávání container uptime dat – prostředí DEV

```
#!/bin/bash

docker ps -a > docker_ps
x=0
cat docker_ps |  while read line;
do
((x++))
awk '{print $NF}' > all-dockers
ps=$(awk '{print $NF}')
done

##########################################

timestamp=$(date +%s)
timerange=310
echo;echo "Time range:  $timerange"
since_timestamp=$(( $timestamp - $timerange ))
echo "Begin time:  $since_timestamp"
echo "Actual time: $timestamp";echo
echo quit | timeout --signal=9 5 docker events --since '5m' > events.log
#"${since_timestamp}000000000"

##########################################

> results
echo "docker ps:"
```

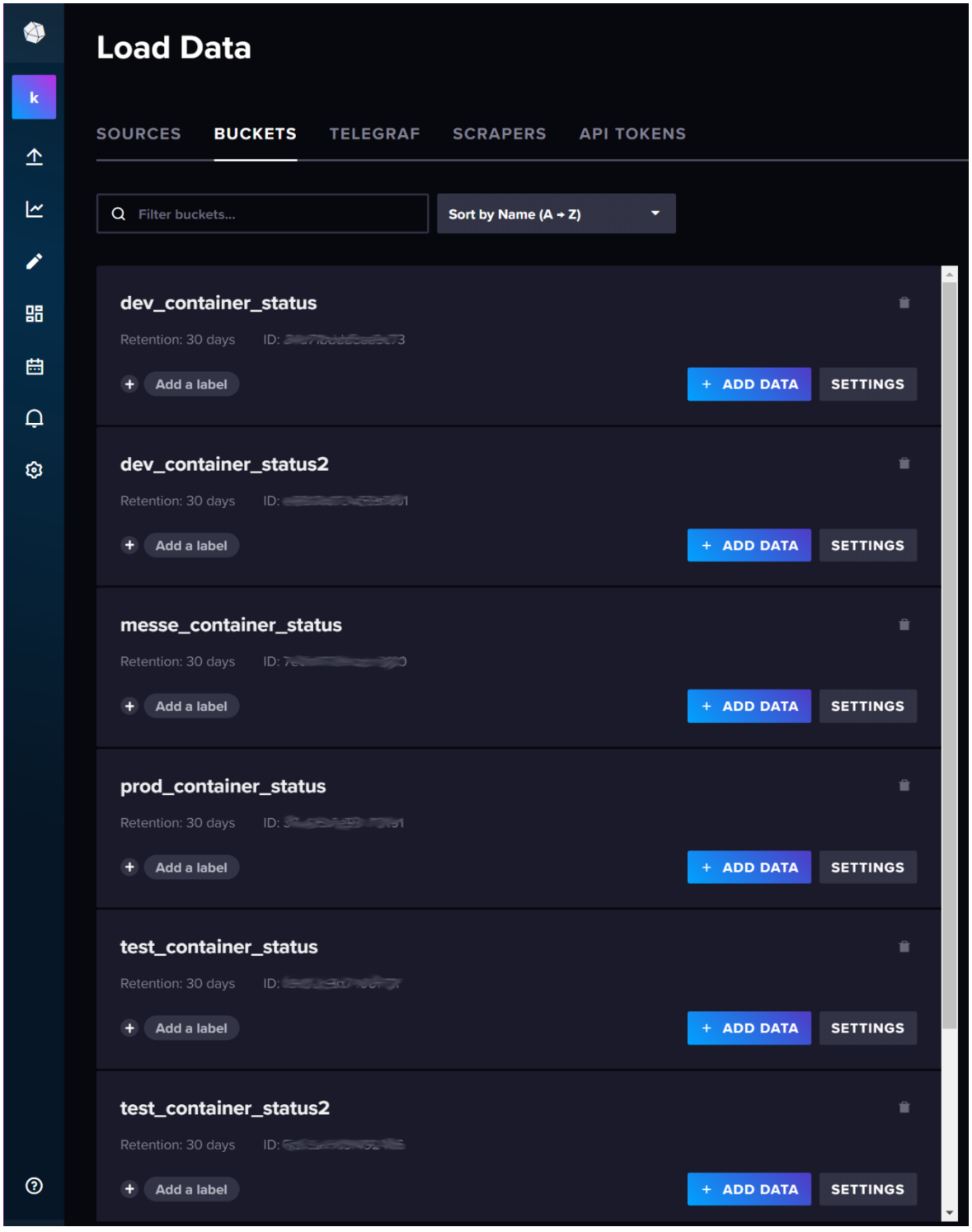
```
y=0
while IFS='=' read -ra line; do
((y++))
  ps=ps_${y}
  container=$(cat all-dockers | sed -n "${y}p")
  declare "$ps"="$container"
echo "    $container"

  status_all=status_${y}
  value_all=1
  declare "$status_all"="$value_all"

echo "$container $value_all" >> results
done < all-dockers
echo

##########################################

> names
> stop_line
i=0
cat events.log |  while read line;
do
if [[ ${line} = *"stop"* ]]; then
((i++))
echo "$line" | grep -oP 'name[^[:blank:]]*' | sed -n '1p' | sed -e
"s/^name=//" -e "s/,$//" >> names
#echo $i
stop_container_bracket=$(cat names | tail -n 1)
echo $stop_container_bracket
if [[ $stop_container_bracket == *")" ]]
then
stop_container=${stop_container_bracket::-1}
else
stop_container=$stop_container_bracket
fi
echo "stopped container n.${i}: $stop_container"

number_of_var=$(cat -n all-dockers | grep $stop_container | awk '{print
$1}')
echo $number_of_var >> stop_line

search="$stop_container 1"
replace="$stop_container 0"
if [[ $search != "" && $replace != "" ]]; then
sed -i "s/$search/$replace/" results
fi
```

```
fi; done; echo

##########################################

a=0
while IFS='=' read -ra line; do
((a++))
  status_stop=status_$(cat stop_line | sed -n "${a}p")
  value_stop=0
  declare "$status_stop"="$value_stop"
done < stop_line

##########################################

containers_number=$(wc -l all-dockers | awk '{print $1}')
echo "Number of containers: $containers_number";echo
echo "Script for send data:";echo
cp request-template request.sh
i=0
z=$containers_number
until [ $i -ge $z ]
do
((i++))
echo "    containerStatus,name='\$ps_$i' dev='\$status_$i'
'$timestamp'000000000" >> request.sh
done
echo "    '" >> request.sh
cat request.sh
bash /root/status-container/request.sh
```

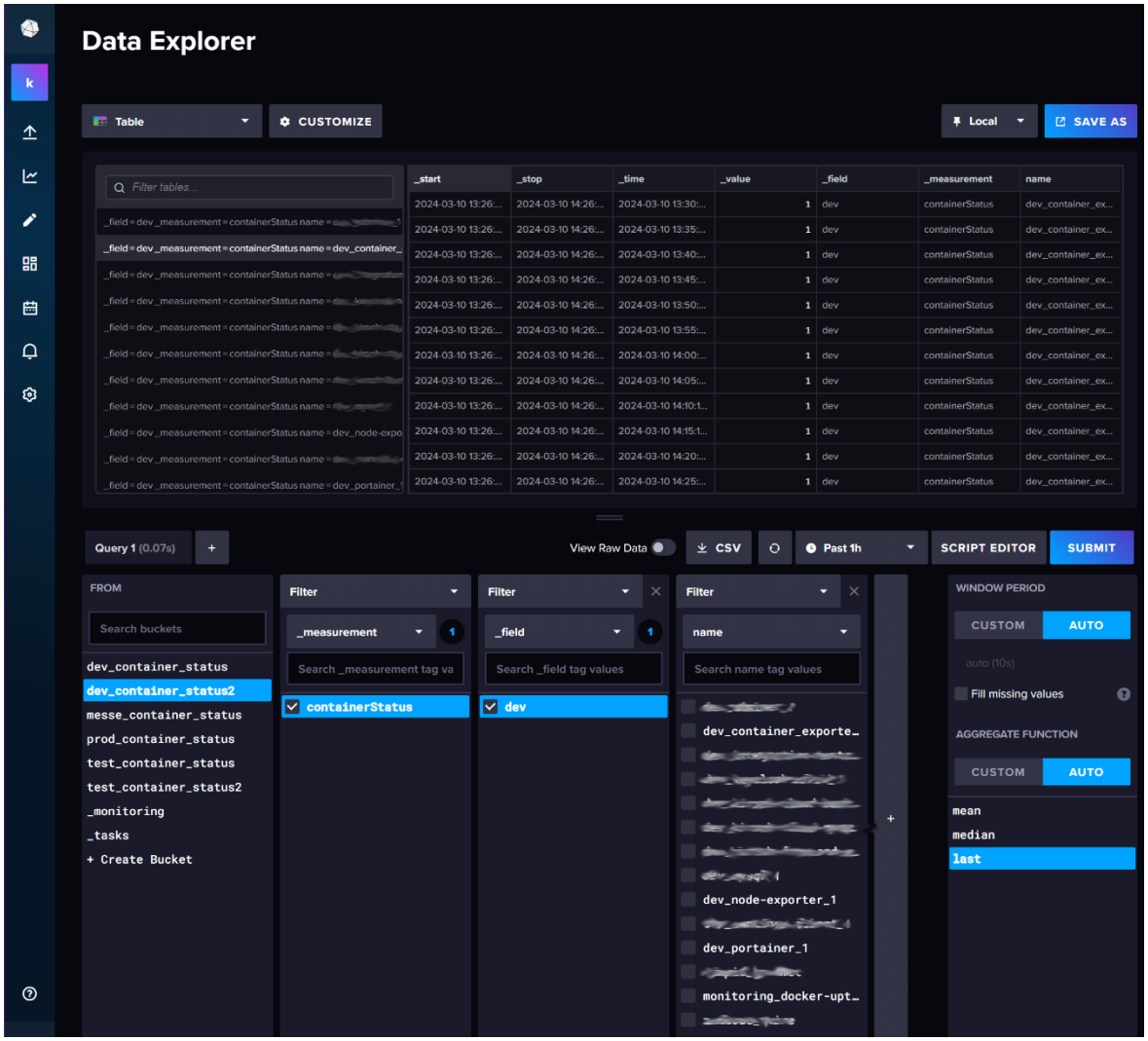
**Příloha L** – Bash script pro získávání container uptime dat – prostředí DEV

```
#!/bin/bash

i=0
while IFS='=' read -ra line; do
((i++))
  ps=ps_${i}
  container=$(cat results | sed -n "${i}p" | awk '{print $1}')
  status=status_${i}
  value=$(cat results | sed -n "${i}p" | awk '{print $2}')
  declare "$ps"="$container"
  declare "$status"="$value"
done < results

curl --request POST \
"http://<IP_DEVOPS_MONITORING>:<PORT>/api/v2/write?org=kirsch&bucket=dev_con
tainer_status2&precision=ns" \
  --header "Authorization: Token <INFLUXDB_API_TOKEN>" \
  --header "Content-Type: text/plain; charset=utf-8" \
  --header "Accept: application/json" \
  --data-binary '
    containerStatus,name='$ps_1' dev='$status_1' '1710023101'000000000
    containerStatus,name='$ps_2' dev='$status_2' '1710023101'000000000
    containerStatus,name='$ps_3' dev='$status_3' '1710023101'000000000
    containerStatus,name='$ps_4' dev='$status_4' '1710023101'000000000
    containerStatus,name='$ps_5' dev='$status_5' '1710023101'000000000
    containerStatus,name='$ps_6' dev='$status_6' '1710023101'000000000
    containerStatus,name='$ps_7' dev='$status_7' '1710023101'000000000
    containerStatus,name='$ps_8' dev='$status_8' '1710023101'000000000
    containerStatus,name='$ps_9' dev='$status_9' '1710023101'000000000
    containerStatus,name='$ps_10' dev='$status_10' '1710023101'000000000
    containerStatus,name='$ps_11' dev='$status_11' '1710023101'000000000
    containerStatus,name='$ps_12' dev='$status_12' '1710023101'000000000
    containerStatus,name='$ps_13' dev='$status_13' '1710023101'000000000
    containerStatus,name='$ps_14' dev='$status_14' '1710023101'000000000
    '
```

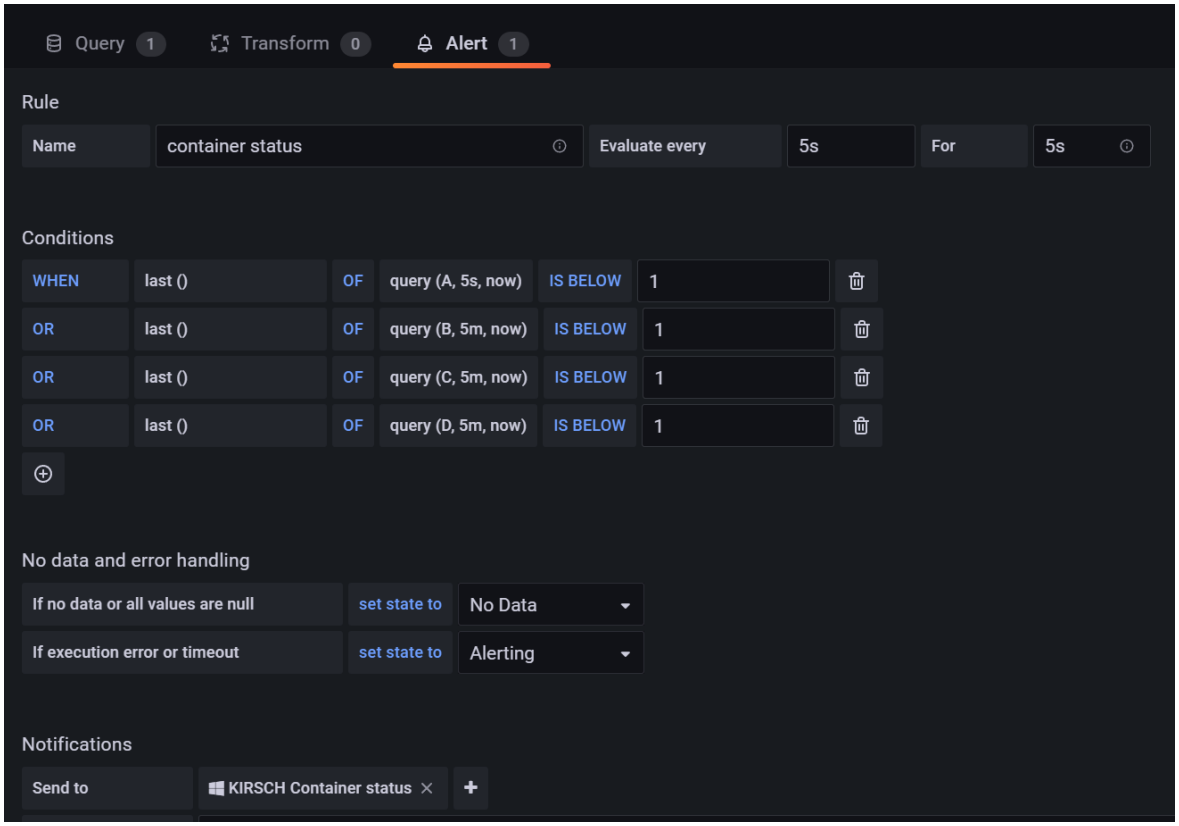
**Příloha M** – Přehled bucketů InfluxDB v organizaci Kirsch



**Příloha N** – Bash script pro získávání container uptime dat – prostředí DEV



**Příloha O** – Nastavení alertů container uptime u všech prostředí Kirsch

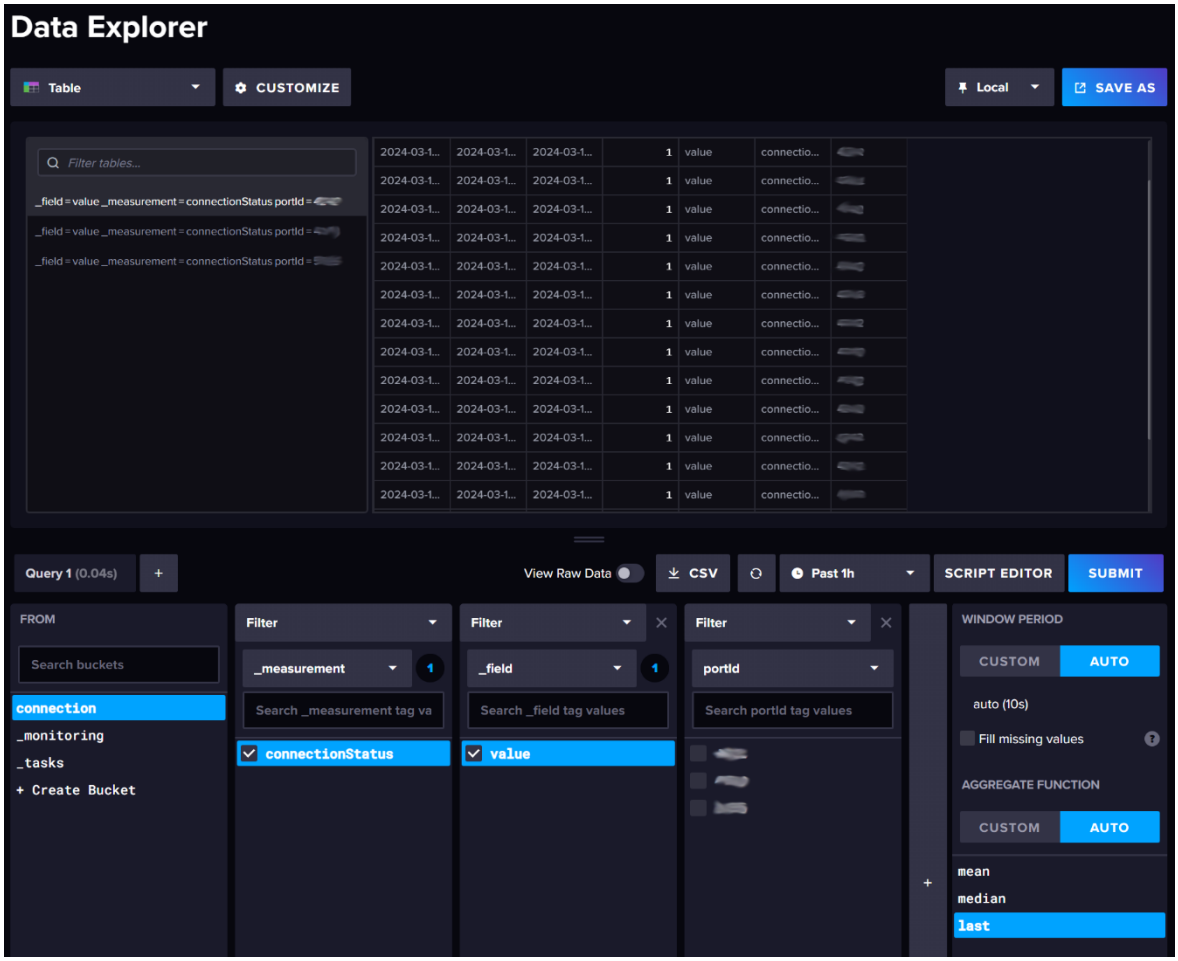



**Příloha P** – Bash script pro získávání dat uptime portů aplikací klienta Comtac

```
#!/bin/bash

timestamp=$(date +%s)
echo $timestamp

echo quit | timeout --signal=9 2 telnet <IP_COMTAC_SERVER> <PORT1> >
<PORT1>.txt
echo quit | timeout --signal=9 2 telnet <IP_COMTAC_SERVER> <PORT2> >
<PORT2>.txt
echo quit | timeout --signal=9 2 telnet <IP_COMTAC_SERVER> <PORT3> >
<PORT3>.txt

connect<PORT1>=$(cat <PORT1>.txt | sed -n '2p' | awk '{print $1}')
echo $connect<PORT1>

connect<PORT2>=$(cat <PORT2>.txt | sed -n '2p' | awk '{print $1}')
echo $connect<PORT2>

connect<PORT3>=$(cat <PORT3>.txt | sed -n '2p' | awk '{print $1}')
echo $connect<PORT3>
```

```
if [ "$connect<PORT1>" == "Connected" ]
then
status<PORT1>=1
else
status<PORT1>=0
fi
echo $status<PORT1>

if [ "$connect<PORT2>" == "Connected" ]
then
status<PORT2>=1
else
status<PORT2>=0
fi
echo $status<PORT2>

if [ "$connect<PORT3>" == "Connected" ]
then
status<PORT3>=1
else
status<PORT3>=0
fi
echo $status<PORT3>

curl --request POST \
"https://influx.devops-
monitoring.com/api/v2/write?org=comtac&bucket=connection&precision=ns" \
  --header "Authorization: Token <INFLUXDB_API_TOKEN>\
  --header "Content-Type: text/plain; charset=utf-8" \
  --header "Accept: application/json" \
  --data-binary '
    connectionStatus,portId=<PORT1> value='$status<PORT1>'
'$timestamp'000000000
    connectionStatus,portId=<PORT2> value='$status<PORT2>'
'$timestamp'000000000
    connectionStatus,portId=<PORT3> value='$status<PORT3>'
'$timestamp'000000000
    '

if [[ "$status_all" == *"0"* ]]
then
echo "restart"
ssh root@<IP_COMTAC_SERVER> 'bash -s < /root/restart-ems-port.sh'
#bash check-ems-port.sh
fi
```

**Příloha Q** – Přehled dat v bucketu pro uptime portů aplikací klienta Comtac



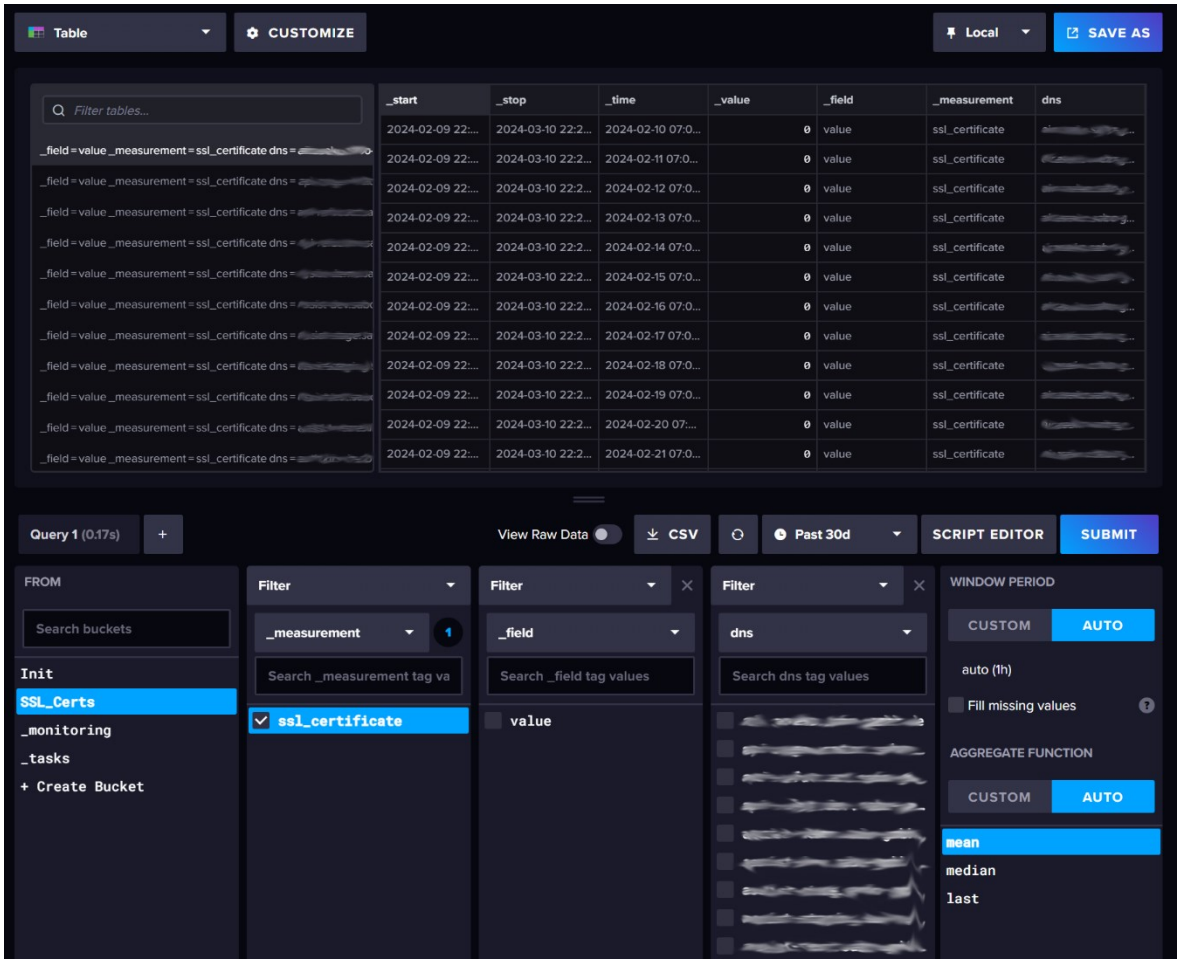
**Příloha R** – Bash script pro získávání dat a přiřazování hodnot expiracím SSL certifikátů

```
#!/bin/bash

path="/root/scripts/ssl_expiration"

timestamp=$(date +%s)
i=0
while IFS='=' read -ra line; do
((i++))
domain_space=$(cat ${path}/.env-domains | sed -n "${i}p")
expire=$(echo quit | timeout --signal=9 3 curl https://$domain_space -vI --
stderr - | grep "expire date" | cut -d":" -f 2-)

domain="$(echo -n "$domain_space" | sed 's/\s*$//')"
domain_num=domain_${i}
declare "$domain_num"="$domain"

if [[ -z ${expire} ]]
then
status=status_${i}
value=7
declare "$status"="$value"

echo "$domain
  - Have some error and is not set
  - status:                  $value
"
else

time=$(echo $expire | awk '{print $3}')
day=$(echo $expire | awk '{print $2}')
month=$(echo $expire | awk '{print $1}')
year=$(echo $expire | awk '{print $4}')

if [[ ${#day} -eq 1 ]]; then day=0$day ; fi
case $month in
    Jan) MON="01" ;;
    Feb) MON="02" ;;
    Mar) MON="03" ;;
    Apr) MON="04" ;;
    May) MON="05" ;;
    Jun) MON="06" ;;
    Jul) MON="07" ;;
    Aug) MON="08" ;;
    Sep) MON="09" ;;
    Oct) MON="10" ;;
```

```
    Nov) MON="11" ;;
    Dec) MON="12" ;;
esac
expire_format=$year"-"$MON"-"$day" "$time
expire_timestamp=$(date -d "$expire_format GMT" +%s)

((time_left=expire_timestamp-timestamp))

status=status_${i}

timestamp_10=864000
timestamp_5=432000
timestamp_3=259200
timestamp_2=172800
timestamp_1=86400
timestamp_0=0

if [ $time_left -ge $timestamp_10 ]
then
  value=0
elif [ $time_left -ge $timestamp_5 ]
then
  value=1
elif [ $time_left -ge $timestamp_3 ]
then
  value=2
elif [ $time_left -ge $timestamp_2 ]
then
  value=3
elif [ $time_left -ge $timestamp_1 ]
then
  value=4
elif [ $time_left -ge $timestamp_0 ]
then
  value=5
else
  value=6
fi

declare -i "$status"="$value"
measurement="ssl_certificate"
field="dns"
value="value"

echo "$domain
  - SSL cert expires in     $expire
  - format:                  $expire_format
```

```
  - timestamp:               $expire_timestamp
  - timestamp untill expire: $time_left
  - status:                  $value
"

fi

done < ${path}/.env-domains

curl --request POST \
"https://influx.devops-
monitoring.com/api/v2/write?org=Sabo&bucket=SSL_Certs&precision=ns" \
  --header "Authorization: Token <INFLUXDB_API_TOKEN>\
  --header "Content-Type: text/plain; charset=utf-8" \
  --header "Accept: application/json" \
  --data-binary "
  $measurement,${field}=${domain_1} ${value}=${status_1}
${timestamp}000000000
  $measurement,${field}=${domain_2} ${value}=${status_2}
${timestamp}000000000
  $measurement,${field}=${domain_3} ${value}=${status_3}
${timestamp}000000000
  $measurement,${field}=${domain_4} ${value}=${status_4}
${timestamp}000000000
  $measurement,${field}=${domain_5} ${value}=${status_5}
${timestamp}000000000
  $measurement,${field}=${domain_6} ${value}=${status_6}
${timestamp}000000000
  $measurement,${field}=${domain_7} ${value}=${status_7}
${timestamp}000000000
  $measurement,${field}=${domain_8} ${value}=${status_8}
${timestamp}000000000
  $measurement,${field}=${domain_9} ${value}=${status_9}
${timestamp}000000000
  $measurement,${field}=${domain_10} ${value}=${status_10}
${timestamp}000000000
#“a další řádky s daty pro curl request”
  "
```

**Příloha S** – Přehled dat v bucketu pro expiraci SSL certifikátů




**Příloha T** – Flux query pro nadefinování dashboardu expirací SSL certifikátů

```
from(bucket: "SSL_Certs")
    |> range(start: -7d)
    |> filter(fn: (r) => r._measurement == "ssl_certificate")
    |> yield(name: "dns")
    |> group(columns: ["dns"])
    |> distinct(column: "dns")
```

**Příloha U** – Konfigurační soubor nástroje Loki *local-config.yml*

```
auth_enabled: false

server:
  http_listen_port: <PORT>
  log_level: error

ingester:
  lifecycler:
    address: 127.0.0.1
    ring:
      kvstore:
        store: inmemory
      replication_factor: 1
    final_sleep: 0s
  chunk_idle_period: 5m
  chunk_retain_period: 30s

schema_config:
  configs:
    - from: 2018-04-15
      store: boltdb
      object_store: filesystem
      schema: v9
      index:
        prefix: index_
        period: 168h

storage_config:
  boltdb:
    directory: /tmp/loki/index

  filesystem:
    directory: /tmp/loki/chunks

limits_config:
  enforce_metric_name: false
  reject_old_samples: true
  reject_old_samples_max_age: 168h
  ingestion_rate_mb: 1024
  ingestion_burst_size_mb: 1024

chunk_store_config:
  max_look_back_period: 0

table_manager:
```

```
  chunk_tables_provisioning:
    inactive_read_throughput: 0
    inactive_write_throughput: 0
    provisioned_read_throughput: 0
    provisioned_write_throughput: 0
  index_tables_provisioning:
    inactive_read_throughput: 0
    inactive_write_throughput: 0
    provisioned_read_throughput: 0
    provisioned_write_throughput: 0
  retention_deletes_enabled: false
  retention_period: 0
```

**Příloha V** – Konfigurační soubor nástroje Promtail *prometheus.yml* pro prostředí Kirsch DEV

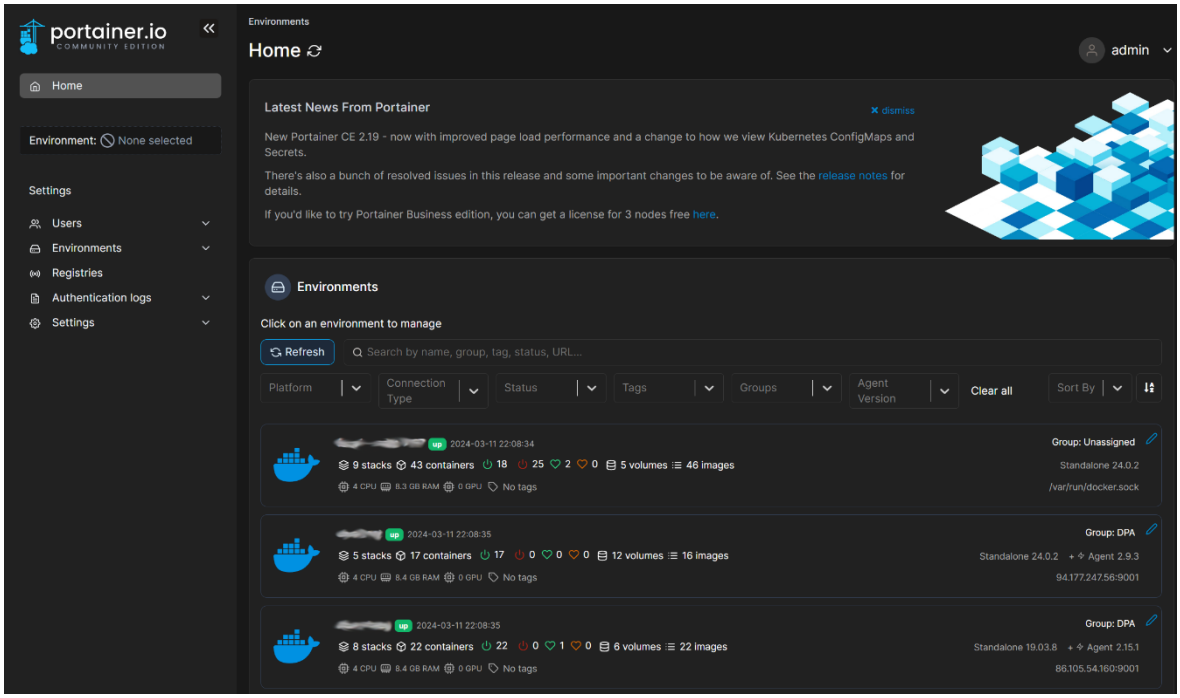
```
server:
  http_listen_address: 0.0.0.0
  http_listen_port: <PORT>

positions:
  filename: /tmp/positions.yaml

clients:
  - url: http://<IP_ADRESA_LOKI>:<PORT>/loki/api/v1/push

scrape_configs:
- job_name: KIRSCH.DEV_console_BE
  entry_parser: raw
  static_configs:
  - targets:
      - localhost
    labels:
      job:  KIRSCH.DEV_console_BE
      cluster: multipass-cluster
      __path__: /logs/belogs/*log

- job_name: KIRSCH.DEV_console_FE
  entry_parser: raw
  static_configs:
  - targets:
      - localhost
    labels:
      job:  KIRSCH.DEV_console_FE
      cluster: multipass-cluster
      __path__: /logs/felogs/*log
```

```
- job_name: KIRSCH-DEV-DB
  entry_parser: raw
  static_configs:
  - targets:
      - localhost
    labels:
      job: KIRSCH-DEV-DB
      cluster: multipass-cluster
      __path__: /logs/dblogs/*log
```

**Příloha W** – Konfigurační soubor Nginx pro nástroj Portainer

```
        server_name portainer.sabo-gmbh.de;

    location / {

        proxy_pass http://127.0.0.1:<PORT>;
        client_max_body_size 200M;


        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
        proxy_set_header X-Forwarded-Host $host;

        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";

        proxy_read_timeout 86400s;
        proxy_send_timeout 86400s;

        }
```

**Příloha X** – Přehled hostů v UI nástroje Portainer




**Příloha Y** – Přehled containerů u vybraného serveru v nástroji Portainer