Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Katedra softwarových technologií

Sledování aplikací pomocí dohledového systému

Bc. Tomáš Kodym

Diplomová práce
2020

Univerzita Pardubice
Fakulta elektrotechniky a informatiky
Akademický rok: 2019/2020

ZADÁNÍ DIPLOMOVÉ PRÁCE
(projektu, uměleckého díla, uměleckého výkonu)

Jméno a příjmení:	**Bc. Tomáš Kodym**
Osobní číslo:	**I18242**
Studijní program:	**N2646 Informační technologie**
Studijní obor:	**Informační technologie**
Téma práce:	**Sledování aplikací pomocí dohledového systému**
Zadávající katedra:	**Katedra softwarových technologií**

Zásady pro vypracování

Cílem diplomové práce je nalézt a nastavit vybraný dohledový systém, který umožní sledování chování nasazených aplikací v podnikovém informačním systému. V teoretické části autor popíše dostupné dohledové systémy, jejich klíčové vlastnosti a přínos pro vyladění podnikových informačních systémů. Praktická část bude obsahovat postup vytvoření vlastního sledovacího schématu pro sledování chování nasazených aplikací v podnikovém informačním systému na vybraném dohledovém systému.

Rozsah pracovní zprávy: **50**
Rozsah grafických prací:
Forma zpracování diplomové práce: **tištěná/elektronická**

Seznam doporučené literatury:

OLUPS, Rihards a Patrik UYTTERHOEVEN. Zabbix 4 Network Monitoring. 3. Packt Publishing, 2019. ISBN 9781789340266.
BASTOS, Joel a Pedro ARAÚJO. Hands-On Infrastructure Monitoring with Prometheus: Implement and scale queries, dashboards, and alerting across machines and containers. Packt Publishing, 2019. ISBN 978-1-78961-234-9.

Vedoucí diplomové práce: **Ing. Soňa Neradová, Ph.D.**
Katedra informačních technologií

Datum zadání diplomové práce: **5. listopadu 2019**
Termín odevzdání diplomové práce: **15. května 2020**

Univerzita Pardubice Česká republika L.S.

Ing. Zdeněk Němec, Ph.D.
děkan

prof. Ing. Antonín Kavička, Ph.D.
vedoucí katedry

V Pardubicích dne 15. listopadu 2019

PROHLÁŠENÍ

Prohlašuji, že jsem tuto práci vypracoval/a samostatně. Veškeré literární prameny a informace, které jsem v práci využil/a, jsou uvedeny v seznamu použité literatury.

Byl/a jsem seznámen/a s tím, že se na moji práci vztahují práva a povinnosti vyplývající ze zákona č. 121/2000 Sb., autorský zákon, zejména se skutečností, že Univerzita Pardubice má právo na uzavření licenční smlouvy o užití této práce jako Školního díla podle § 60 odst. 1 autorského zákona, a s tím, že pokud dojde k užití této práce mnou nebo bude poskytnuta licence o užití jinému subjektu, je Univerzita Pardubice oprávněna ode mne požadovat přiměřený příspěvek na úhradu nákladů, které na vytvoření díla vynaložila, a to podle okolností až do jejich skutečné výše.

Beru na vědomí, že v souladu s § 47b zákona č. 111/1998 Sb., o vysokých školách a o změně a doplnění dalších zákonů (zákon o vysokých školách), ve znění pozdějších předpisů, a směrnicí Univerzity Pardubice č. 9/2012, bude práce zveřejněna v Univerzitní knihovně a prostřednictvím Digitální knihovny Univerzity Pardubice.

V Pardubicích dne Bc. Tomáš Kodym

PODĚKOVÁNÍ:

Zde bych chtěl poděkovat paní Ing. Soně Neradové Ph.D. za strávený čas, cenné rady a důležité poznatky při zpracování diplomové práce. Dále bych chtěl poděkovat celé své rodině a přítelkyni za poskytnutou oporu během celého studia.

ANOTACE

Tato práce se zabývá problematikou týkající se moderních dohledových systémů. V práci jsou popsány populární dohledové systémy včetně jejich architektury. Záměrem diplomové práce je nalézt a nastavit vybraný dohledový systém, který umožní sledování chování aplikací v podnikovém informačním systému.

KLÍČOVÁ SLOVA

Prometheus, Grafana, Monitoring, Docker, Docker-Compose, Node-exporter, Cadvisor, Spring Boot

TITLE

Monitoring applications using a monitoring system

ANNOTATION

This thesis deals with the issues related to the modern monitoring systems. The thesis describes popular monitoring systems, including their architecture. The aim of the thesis is to find and set up selected monitoring system, that will allow monitoring of applications in corporate information system.

KEYWORDS

Prometheus, Grafana, Monitoring, Docker, Docker-Compose, Node-exporter, Cadvisor, Spring Boot

OBSAH

SEZNAM TABULEK

SEZNAM OBRÁZKŮ

SEZNAM ZKRATEK

API	Application Programming Interface
CI	Continuous Integration
CD	Continuous Delivery
CPU	Central Processing Unit
ETL	Extract, transform, load
HTTP	Hypertext Transfer Protocol
ICMP	Internet Control Message Protocol
IoT	Internet of Things
IP	Internet Protocol
IT	Informační Technlogie
JDBC	Java Database Connectivity
JMX	Java Management Extention
JSON	JavaScript Object Notation
JVM	Java Virtual Machine
MVC	Model View Controller
OpenJDK	Open Java Development Kit
RAM	Random Access Memory
REST	Representational State Transfer
SMTP	Simple Mail Transfer Protocol
TCP	Transmission Control Protocol
TSDB	Time Series Database
YAML	YAML Ain't Markup Language

ÚVOD

Prostřednictvím monitoringu dochází ke sběru informací o prostředí. Monitorovací systém je navržen tak, aby sbíral potřebné informace průběžně a v reálném čase. Monitorovací systém nejenže je nápomocný pro získání celkového přehledu o stavu prostředí, aktuálně probíhajících procesech nebo výkonu aplikací, ale zároveň jej lze využít pro zvýšení bezpečnosti prostředí. Včasné odhalení hrozby má značný vliv na snížení doby trvání daného incidentu. Přičemž některé problémy nemusí být odhaleny ihned, a nemusí být ani přímo monitorovány, avšak sledování jiných důležitých metrik, může pomoci problém odhalit. Cílem této diplomové práce je nalézt a nastavit vybraný dohledový systém, který umožní sledování chování nasazených aplikací v podnikovém informačním systému.

První část diplomové práce je věnována teoretické části, v níž je nejdříve představena virtualizační technologie Docker, která je v této práci velmi využívaná a pochopení základních principů přispěje k porozumění práci. Následně jsou představeny jednotlivé monitorovací nástroje: Prometheus, Graphite, Zabbix, TICK, ELK a Splunk. Tyto nástroje jsou charakterizovány, je popsán jejich princip, architektura a jsou shrnuty jejich základní výhody a nevýhody.

Praktická část je obsažena v druhé části této diplomové práce. Nejdříve je detailně představeno monitorované prostředí včetně použitých technologií. Následně je předveden postup vytvoření aplikace, která bude spuštěna jako kontejner v technologii Docker. Následuje znázornění nastavení a zprovoznění monitorovacího nástroje Prometheus včetně podrobného popisu konfigurace. Po úspěšném zprovoznění monitorovacího systému Prometheus jsou představeny technologie: Node Exporter, Cadvisor, Docker daemon metrics, které jsou využívány jako exportéry metrik, jež mají být sledovány. V následujících podkapitolách jsou popsány technologie Spring Boot Actuator a Micrometer, které jsou využity pro vytváření vlastních typů metrik využitých ve sledované aplikaci.

V závěru práce je představena vizualizační technologie Grafana. Tento nástroje je využit pro vizualizaci dat získaných monitorovacím nástrojem Prometheus. Je zde předvedeno připojení na monitorovací nástroj, vytváření vizualizačních dashboardů a konfigurace varovných notifikací. Po nakonfigurování celého prostředí je demonstrována ukázka selhání sledovaného systému a předvedení odesílání informativních notifikací.

1. TEORETICKÁ ČÁST

Spravovat vlastní monitorovací systém je aktuálně ve světě plném technologií již nezbytnou součástí poskytování IT služeb. Kvalitně navržený monitorovací systém nejenže zvýší spolehlivost a stabilitu poskytovaných služeb, ale zároveň umožňuje získat celkový přehled o stavu prostředí nebo o aktuálně probíhajících procesech. Data získaná prostřednictvím monitoringu lze dále zpracovávat. Tímto způsobem je možné předejít určitým situacím, které by při absenci monitoringu byly objeveny po delší době. Pod tímto si lze představit vysoké vytížení serveru, síťový problém či docházející místo na disku. (Ellingwood, 2020; *WHY YOU NEED A MONITORING SYSTEM*, 2020)

1.1. Účel monitoringu

Technologie ani lidé nejsou neomylní a chyby při poskytování IT služeb nastávají. Omyly mohou být způsobeny výpadky proudu, síťovými chybami nebo chybami při nasazení aplikace do produkčního prostředí. Může snadno dojít k omylu, který člověk nezaregistroval a aplikace nenastartuje správně. Pokud je monitoring optimálně nastavený, může takovou chybu velmi rychle odhalit a pracovník má šanci svou chybu napravit. Důvodů proč se vyplatí monitorovat prostředí je více (*WHY YOU NEED A MONITORING SYSTEM*, 2020):

- Kontrola využití zdrojů, prevence
    - Administrátor prostředí má přehled o aktuálních stavech zdrojů na komponentách a na jejich základě je schopný analyzovat případný problém a vyřešit jej s předstihem dříve, než dojde k fatální chybě nebo selhání systému.
- Detekce incidentů
    - Umožňuje odhalit nastalý problém nebo incident ve velmi krátkém čase. Tím se snižuje čas, po který by byla služba nedostupná a zároveň zvyšuje její dostupnost a tím pádem i potenciální zisk.
- Časová úspora
    - Monitoring zajistí pravidelnou kontrolu stavu prostředí a aplikací. Jeho zásluhou se mohou zaměstnanci věnovat jiným aktivitám.

- Zabezpečení
    - Monitorovací systém lze využít pro zvýšení zabezpečení prostředí. Lze odhalit podezřelé logování (pokusy o přihlášení), změny v konfiguračních souborech nebo například zvýšený síťový provoz.

1.1.1. Metrika

Metriky si lze představit jako surová data, která jsou získána ze sledované komponenty. Například operační systém vystavuje data o základním využití zdrojů (využití CPU, RAM nebo velikosti místa na disku). Některé další komponenty jako webové nebo databázové servery vystavují své vlastní metriky pro monitorování. V případě vlastních aplikací není metrika nastavena ve výchozím stavu a je potřeba danou funkcionalitu doprogramovat nebo použít dostupnou knihovnu. Metrika je sbírána monitorovacími systémy a následně vyhodnocována. Metriky jsou významné, neboť poskytují pohled na aktuální stav systému. (Ellingwood, 2020)

Typy metrik lze charakterizovat následovně:

- Hostitelské metriky – Tyto metriky slouží pro vyhodnocování aktuálního stavu hostitelského zařízení. Mezi ně lze zařadit například:
    - využití CPU,
    - využití paměti,
    - velikost místa na disku,
    - procesy.
- Aplikační metrika – Tento typ metrik tvoří vrstvu nad hostitelskými metrikami, jejichž zdroje jsou využívány. Tato úroveň metrik je ukazatelem stavu a výkonu nasazených aplikací. Na úrovni aplikační metriky lze monitorovat:
    - nastalé chyby,
    - restart, případně selhání služby,
    - výkon a odezvu,
    - využití aplikačních zdrojů.
- Síťová metrika – Zajišťuje kontrolu stavu komunikace mezi zařízeními nebo případně aktuální využití sítě. Pro monitorování sítě mohou být využity následující typy metrik:

    - stav připojení,
    - ztrátovost,
    - odezva,
    - šířka pásma.
- Serverová metrika – Data nejsou shromažďována pro jednotky hostitelských zařízení. Ale pro celé skupiny serverů. Následně zmíněné metriky mohou sloužit pro získání informací o celých sestavách, což umožňuje získat nadhled o aktuální zátěži a latenci. Případné typy serverových metrik jsou:
    - využití zdrojů,
    - škálovatelnost,
    - nefunkční instance.
- Externě využívané metriky – Pod tímto si lze představit aplikace třetích stran, které jsou využívány pro potřeby projektu. Aplikace většinou pro komunikaci poskytují API (Application Programming Interface), které informuje o výpadku (HTTP kódy), avšak nedostupnost poskytovaných služeb může způsobit problém. Je tedy vhodné monitorovat následující metriky:
    - dostupnost služeb,
    - úspěšnost, popřípadě chybovost přístupů,
    - četnost spouštění.

1.1.2. Monitoring

Monitoring je proces sbírání, zpracovávání a analyzování dat, která jsou získána monitorovacím systémem a vytvářejí celkový obraz aktuálního stavu o sledovaném prostředí. Monitorovací systém je zodpovědný za sbírání, ukládání, vyhodnocování a vizualizaci dat. Získaná data se skládají ze surových dat, která nabývají smyslu až poté, co jsou zpracována, zanalyzována a stávají se z nich informace, které poskytují pohled na stav systému. Metriky lze chápat jako bod, který představuje stav systému v čase. Proto je užitečné zobrazit i předešlé naměřené hodnoty, aby bylo možné zaznamenat aktuální trend. Tato možnost se nazývá vizualizace. (Ellingwood, 2020)

1.1.3. Upozornění

Zasílání upozornění je součást monitorovacího systému zodpovědná za provádění akcí na základě změn naměřených hodnot. Definice upozornění se vždy skládá ze dvou komponent:

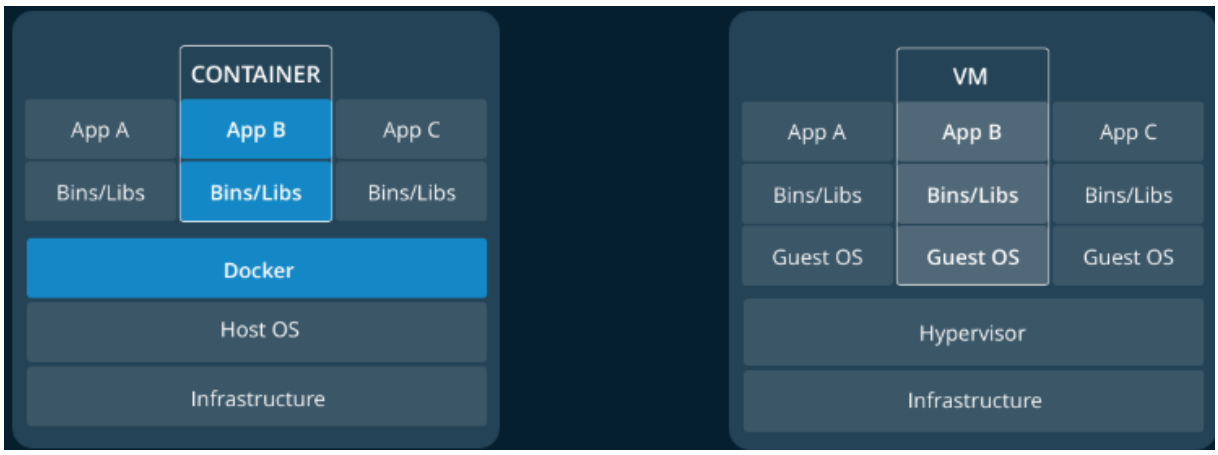
- práh – hodnota, při jejímž překročení je provedena akce,
- akce – která se provede po překročení dané hodnoty.

Hlavním účelem upozornění je varovat o změně stavu monitorovaných systémů. Upozornění by mělo vždy obsahovat detailní informace o nastalém problému (kde problém nastal, kdy problém nastal, popřípadě koho kontaktovat). (Ellingwood, 2020)

1.2. Docker

Vzhledem k tomu, že v tomto projektu je technologie Docker mnohokrát používána, jsou v následující kapitole představeny základní pojmy a architektura potřebné pro porozumění dané problematice. Později v této práci jsou využívané monitorovací nástroje a aplikace, které jsou nasazeny a posléze spuštěny právě v Docker kontejnerech.

Technologie Docker je poměrně mladá, ale celosvětově již velmi dobře známá, open source platforma využívaná pro virtualizaci. Hlavní doménou platformy Docker je podpora vývoje, přenositelnosti a provozu aplikací. Pro virtualizaci jsou používány na sobě nezávislé a navzájem izolované kontejnery. Izolace a zabezpečení kontejnerů umožňuje na sobě nezávislé spouštění velkého množství různorodých kontejnerů na hostitelském operačním systému. Na rozdíl od virtuálního stroje, který pro svůj běh musí využívat technologii hypervisor, jenž také navíc zatěžuje operační systém, běží kontejnery přímo v jádře hostitelského operačního systému. Tento přístup umožňuje běh i několika desítek kontejnerů na jednom hostitelském zařízení. (*Docker overview,* 2019)



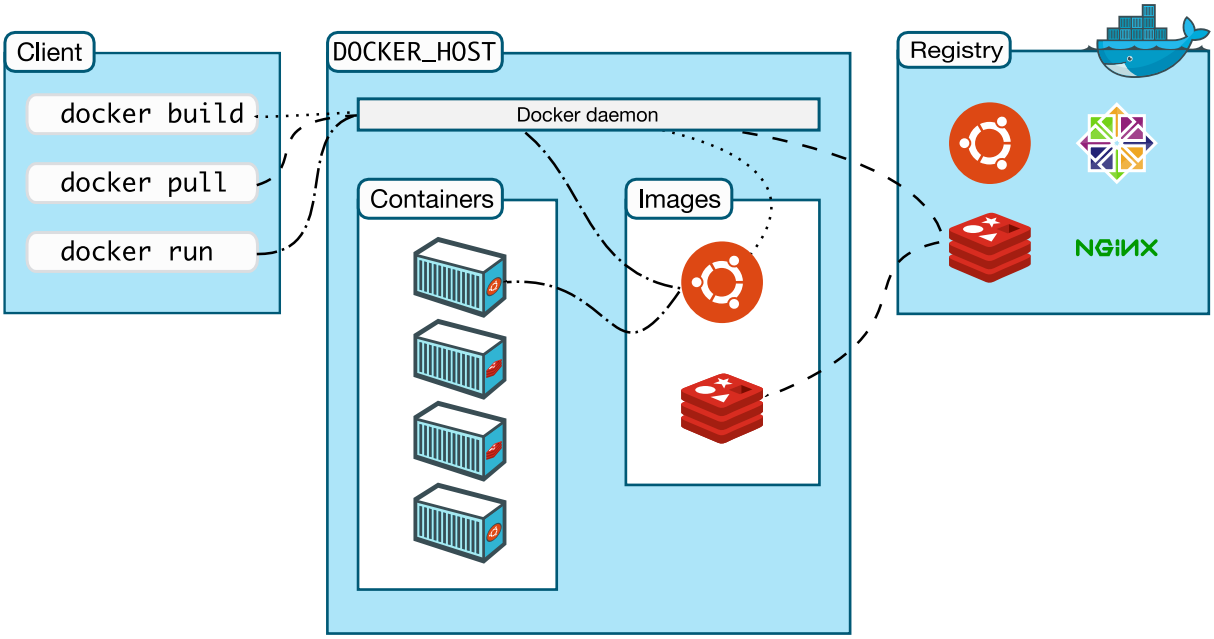
Obrázek 1 – Docker vs. Virtualizace

Zdroj: (Orientation and setup, 2019)

Na uvedeném obrázku (Obrázek 1) je zobrazeno grafické znázornění rozdílu mezi technologií Docker a klasickým virtuálním strojem. Na první pohled je zřejmý rozdíl v architektuře. Kontejnery v Docker architektuře sdílí jádro operačního systému. Každý kontejner se tedy chová pouze jako běžící proces a v daný moment není nutná nadbytečná režie, která by byla potřebná pro běh klasického virtuálního stroje. Tato technologie je odlišná od virtuálního stroje především využitelností zdrojů hostitelského zařízení, které mohou být potřebné pro jiné důležitější úlohy. (*Orientation and setup*, 2019)

1.2.1. Docker architektura

V této podkapitole je představena základní architektura technologie Docker nutná pro pochopení základních vztahů mezi komponentami.



Obrázek 2 – Docker architektura

Zdroj: (Docker overview, 2019)

Technologie Docker využívá architekturu klient-server. Schéma této architektury je znázorněno na výše uvedeném obrázku (Obrázek 2). Docker deamon komunikuje s Docker klientem, který mu zasílá požadavky za pomoci REST API, UNIX soketů nebo síťového rozhraní. (*Docker overview*, 2019)

- Docker daemon – Neboli „dockerd“, je proces, který je zodpovědný za správu Docker komponent, kterými jsou images, kontejnery, sítě a úložiště.
- Docker client – Slouží pro interakci mezi uživatelem a Docker démonem. Uživatel komunikuje s Docker klientem, který přes Docker API předává instrukce na Docker démona, který je následně zpracuje.
- Docker registry – Jsou využívány jako úložiště pro Docker images. V tomto projektu bude dále využito úložiště Docker Hub, které je veřejně dostupné a může ho používat kdokoliv.
- Docker images – Šablona, která obsahuje instrukce pro vytvoření Docker kontejneru. Image tedy funguje jako předpis pro vytváření kontejneru. Z jednoho image lze vytvořit několik různých na sobě nezávislých kontejnerů. Images je možné mezi sebou libovolně kombinovat.

- Lze také vytvářet vlastní návrhy image, které je možné ukládat na Docker registry. V praktické části tohoto projektu je předvedeno vytváření Docker images pomocí Dockerfile.

- Docker containers – Kontejner je spustitelná instance Docker image. Nejedná se o nic jiného než o běžící proces aplikace. Kontejner je vždy vytvořen vůči Docker image. V praxi to znamená, že jakákoliv změna v běžícím kontejneru nebude uložena, pokud bude kontejner odstraněn a opětovně vytvořen.

1.2.1. Motivace pro využití technologie Docker

Využívání technologie Docker při vývoji softwaru zjednodušuje cyklus vývoje. Princip kontejnerů je velmi vhodný pro práci s nástroji podporujícími průběžnou integraci neboli CI/CD, kdy nová verze aplikace od vývojáře je okamžitě nasazena na testovací prostředí. Po ověření funkčnosti na testovacím prostředí může být aplikace nasazována dále, prostřednictvím uživatelského akceptačního prostředí až na produkční prostředí k zákazníkovi. Velkou výhodou kontejnerů je nezávislost na zvoleném prostředí. Kontejnery mohou být spuštěny na lokálním počítači vývojáře, na virtuálním počítači, v datovém centru anebo v cloudu. Nenáročnost a přenositelnost technologie Docker z ní vytváří ideální náhradu za virtuální stroje, vzhledem k jednoduchosti, přenositelnosti a úspoře zdrojů. (*Docker overview,* 2019)

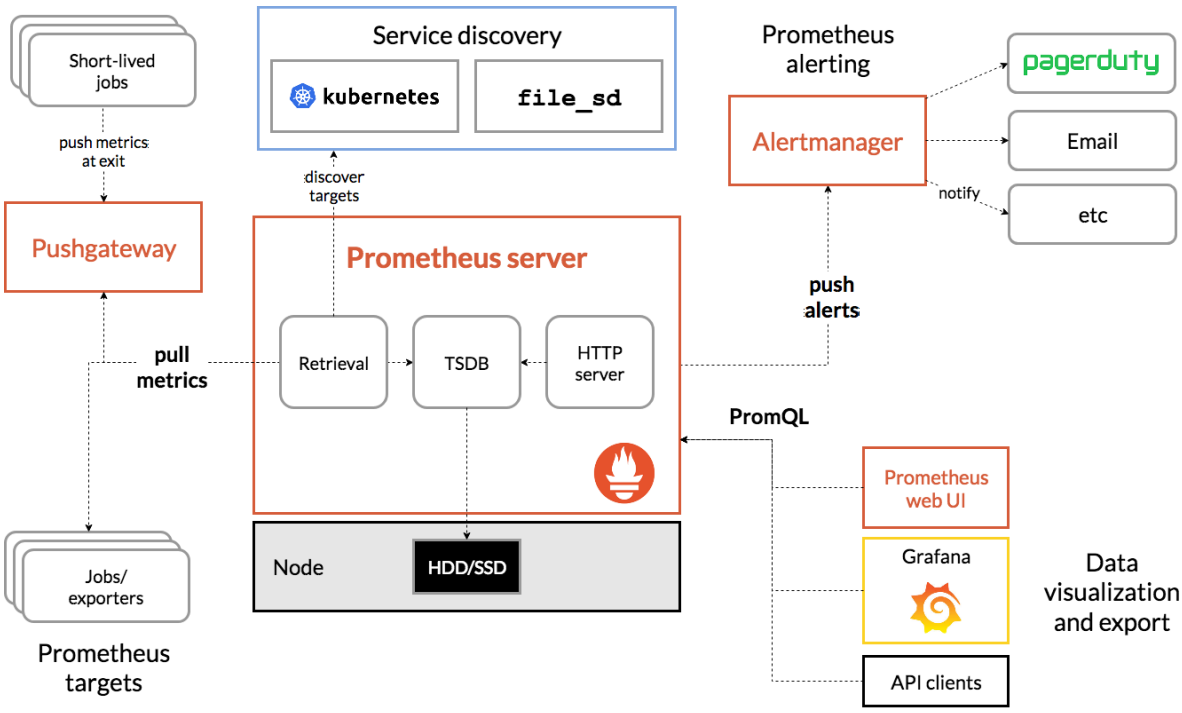
1.3. Monitorovací nástroje

V následujících podkapitolách je popsáno několik aktuálně nejpoužívanějších monitorovacích nástrojů. U vybraných nástrojů je nastíněna jejich architektura, případně topologie a jsou popsány jejich nejvýraznější výhody a nevýhody. Shrnutí základních poznatků o vybraných monitorovacích nástrojích by mělo vést k vytvoření přehledu o některých dostupných monitorovacích nástrojích a možnosti jejich využití.

1.4. Prometheus

Nástroj Prometheus je Open source monitorovací systém, který slouží pro monitorování událostí, spouštění agregačních operací nad sesbíranými daty, vyhodnocování monitorovacích pravidel a následné zasílání upozorňovacích/výstražných notifikací. Tento nástroj se pro svoji jednoduchost, nenáročnost na použití, velké množství možných integrací a podporu komunity, stal jedním z nejvyužívanějších monitorovacích nástrojů. Samozřejmostí je zakomponované jednoduché a přehledné grafické uživatelské rozhraní, které umožňuje tvorbu grafů, úpravu konfigurací a vytváření pravidel. (*Overview,* 2020)

Pro naprogramování tohoto systému byl využit programovací jazyk GO, který vzhledem ke svým vlastnostem, zaručuje velmi vysoký výkon vyhledávání v nasbíraných datech. Prometheus se řadí mezi takzvané time series monitorovací systémy. V praxi to znamená, že všechna získaná data jsou opatřena časovým razítkem. Stahování metriky, která je vystavena v monitorovaných cílech, je prováděno přes HTTP. Jeden cíl souvisí vždy pouze s jedním procesem nebo aplikací. Na následujícím obrázku (Obrázek 3) je zobrazena architektura monitorovacího nástroje Prometheus.



Obrázek 3 – Prometheus architektura

Zdroj: (Overview, 2020)

1.4.1. Prometheus architektura

V následující části je podrobněji popsána architektura monitorovacího nástroje Prometheus.

- Prometheus Server (*Prometheus*, 2020)
    - Retrieval – Využívá se pro získávání dat z monitorovaných cílů.
        - Data mohou být získávána přímo z monitorovaných cílů nebo z komponenty Pushgateway.
    - TSDB – Time Series Database – Slouží pro ukládání dat, která jsou opatřena časovým razítkem.
        - Prometheus poskytuje vlastní lokální databázi pro ukládání časově uspořádaných dat.
        - Prometheus poskytuje rozhraní, které je možné využít pro integraci s jinými databázemi.
- Pushgateway – Tuto komponentu si lze představit jako prostředníka, který pracuje jako spojka mezi krátkodobými procesy a Prometheem. (*Prometheus Pushgateway*, 2020)

    - Prometheus ve výchozím nastavení získává data pomocí „pull“ metody, kdy si sám podle konfigurace určí, kdy má data stáhnout. Nicméně, některé úlohy nebo procesy mohou mít tak krátký průběh, že nemusí být zastiženy při běhu.
    - Nabízí se možnost řešení, využít po doběhnutí procesu takzvanou „push“ metodu, kdy dobíhající proces pouze poskytne data komponentě Pushgateway, která je následně vystaví pro Prometheus.
- Exporters – Jedná se o nástroje třetích stran, jejichž úkolem je získat informace o běžící aplikaci a následně tuto informaci zmodifikovat takovým způsobem, aby byla dostupná ve správném formátu pro další zpracování. Většina exportérů je dostupná ke stažení či využití jako Docker kontejnery. Některé zajímavé exportéry jsou vypsány níže (*EXPORTERS AND INTEGRATIONS*, 2020):
    - cAdvisor,
    - MySQL server exporter,
    - Node/system metrics exporter,
    - JMX (Java Management Extention) exporter.
- Alertmanager – Komponenta, kterou si lze představit jako prostředníka mezi monitorovacím systémem a uživatelem. Alertmanager, jak už z názvu vyplývá, slouží pro zpracovávání a následný management varovných notifikací. Notifikace jsou poté distribuovány podle konfigurace dál. Některé podporované komunikační kanály jsou vypsány níže (*Alertmanager*, 2020):
    - Email,
    - HipChat,
    - PagerDuty,
    - Slack,
    - Webhook.
- Service Discovery – Nástroj, který je využíván pro automatickou detekci komponent, které mají být monitorovány. Tímto způsobem se výrazně sníží potřeba statického konfigurování. Aktuálně jsou dvě možnosti, jak nakonfigurovat monitoring pomocí Service Discovery. (*Implementing Custom Service Discovery*, 2020)

- File based Service Discovery
  - V podstatě se jedná o soubor, který obsahuje 1-n statických konfigurací, které byly vygenerované nějakou aplikací, která zanalyzovala vystavěné prostředí a získala informaci o běžících komponentách.
    - Zřejmou nevýhodou tohoto přístupu je nutnost implementace vlastní aplikace pro analýzu prostředí.
- Service Discovery – podporované služby:
  - Amazon Web Services – AWS SD umožňuje načítání cílů k monitorování z AWS EC2.
  - Open Stack – OpenStack SD umožňuje načítání cílů k monitorování z instancí OpenStack Nova.
  - Kubernetes – Kubernetes SD umožňuje načíst cíle z REST API Kubernetes.

V následující části je provedeno shrnutí kladných a záporných vlastností monitorovacího systému Prometheus. (*COMPARISON TO ALTERNATIVES*, 2020)

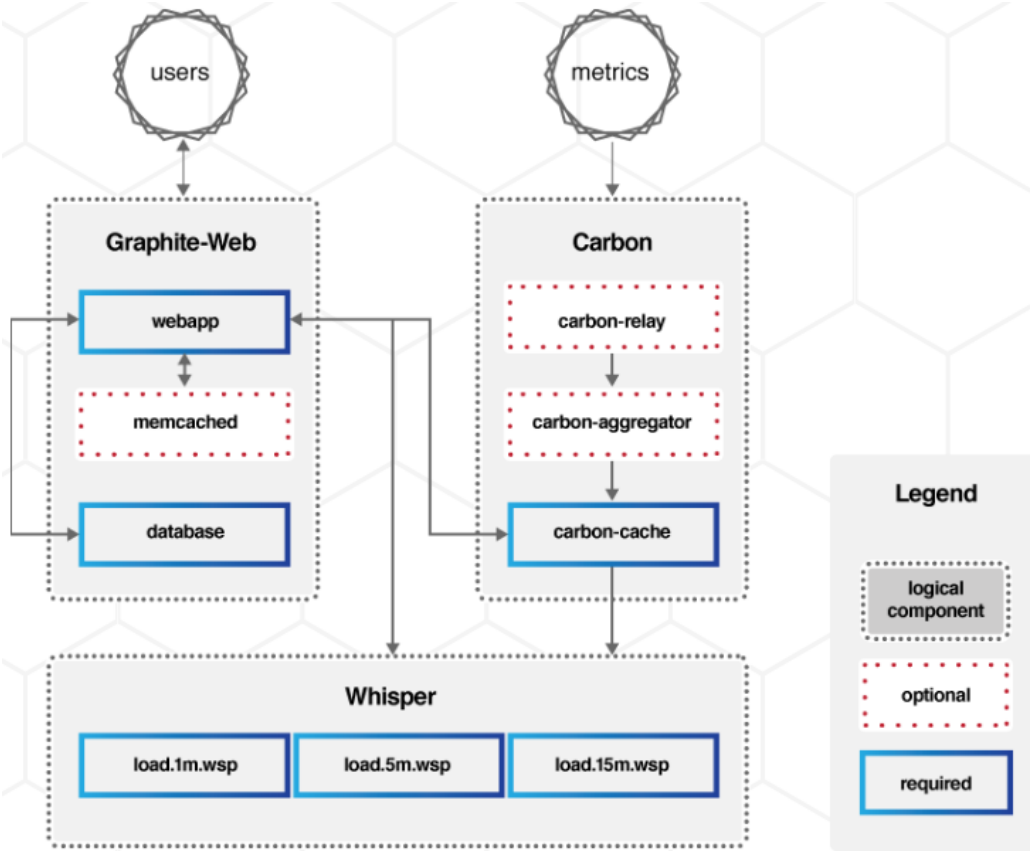
- Výhody:
  - jednoduché ovládání,
  - rychlá instalace,
  - výkonný dotazovací jazyk PromQL,
  - velký výběr exportérů pro monitoring mnoha druhů aplikací,
  - zdarma,
  - velká podpora komunity,
  - dostupné na Docker Hub.
- Nevýhody:
  - Prometheus nepodporuje ukládání a následnou analýzu strojových dat (logů),
  - sběr a práce s nenumerickými a strukturovanými daty,
  - nevhodný pro dlouhodobé skladování dat (výchozí nastavení je 15denní retenční čas).

1.5. Graphite

Monitorovací nástroj Graphite je monitorovací systém naprogramovaný v jazyce Python, který pro spuštění a provoz nevyžaduje nákladný a výkonný hardware. Graphite aktivně nesbírá data, ale pouze pasivně čeká na data, která je nutné mu poskytnout externím nástrojem. Monitorovací aplikace musí být nakonfigurována tak, aby zasílala data přímo do Graphite. Slouží tedy výhradně pro vykreslování již uložených časově uspořádaných dat. (*Overview*, © 2011-2017)

1.5.1. Graphite architektura

Pro získání ucelené představy je architektura monitorovacího nástroje Graphite předvedena na následujícím obrázku (Obrázek 4).



Obrázek 4 – Graphite architektura

Zdroj: (Overview, © 2011-2017)

Z uvedeného obrázku vyplývá, že monitorovací nástroj Graphite se dělí na tři hlavní části, které se starají o příjem, zpracování a následný zápis dat. Součásti budou podrobněji rozebrány níže:

- Carbon – Služba sloužící primárně pro příjem dat a přípravu dat k uložení. Dělí se na 3 další části/démony, z nichž dva jsou nepovinné. (*The Carbon Daemons*, © 2011-2017)
    - carbon-cache – Komponenta zodpovědná za příjem dat a následně jejich okamžitý zápis. Data jsou nejdříve nahrávána do paměti cache, a pokud je jich dostatečné množství, jsou jako jedna dávka pomocí komponenty Whisper zapsána na disk.
    - carbon-relay – Nepovinná součást, pomocí které je možné ukládat data dvěma způsoby:
        - Replikace dat – Zápis příchozích dat na dvě nebo více úložišť jako duplikát.
        - Dělení dat – Třídění příchozích dat podle nastavených pravidel a následný zápis na dvě nebo více úložišť. Tato technologie je určena pro vysokou škálovatelnost a rychlost vyhledávání dat.
    - carbon-aggreagator – Nepovinná součást, kterou je možné využít pro snížení počtu vstupně/výstupních operací pomocí agregování příchozích dat.
- Whisper – Databáze založená na principu Round-Robin databáze. Poskytuje rychlé a spolehlivé ukládání dat. Z obrázku č. 4 je zřejmé, že data jsou ukládána do více archivů podle specifického časového intervalu. (*The Whisper Database*, © 2011-2017)
- Graphite-web – Je grafické uživatelské rozhraní pro načítání dat z Whisper databáze a jejich následné zobrazování. (*Webapp Database Setup*, © 2011-2017)
    - Database – SQLite databáze určena pro ukládání uživatelských účtů, nastavení a dashboardů.
    - Web-app – Poskytuje funkce pro základní vizualizace dat.

V následující části je proveden souhrn kladných a záporných vlastností monitorovacího systému Graphite. (Berman, 2020; *COMPARISON TO ALTERNATIVES*, 2020)

- Výhody:
    - persistence dlouhodobých dat,
    - velké množství nástrojů pro sběr dat,

- vysoká škálovatelnost.

- Nevýhody:
  - pasivní sběr dat,
  - malé množství nástrojů pro sběr dat z kontejnerových aplikací,
  - absence varovných notifikací.

1.6. Zabbix

Zabbix je open source monitorovací nástroj, který nabízí komplexní dohled nad aplikačními komponentami. Lze si pod tím představit aplikace, servery, síťové prvky, virtuální stroje nebo kontejnery. Grafické rozhraní je naprogramované v jazyce PHP, zatímco backend je naprogramován v jazyce C. Zabbix poskytuje přehlednou vizualizaci dat a sofistikovanou možnost reportování událostí. Tato vlastnost ukrývá například eskalaci na nadřízeného zaměstnance v případě absence reakce na reportovaný problém. Monitoring je provozován pomocí vlastních agentů, které je možné nainstalovat na většinu moderních operačních systémů nebo pomocí protokolu SNMP, TCP nebo ICMP. (*Distributed Monitoring*, © 2001-2020; *Notifications & Automatic actions*, © 2001-2020)

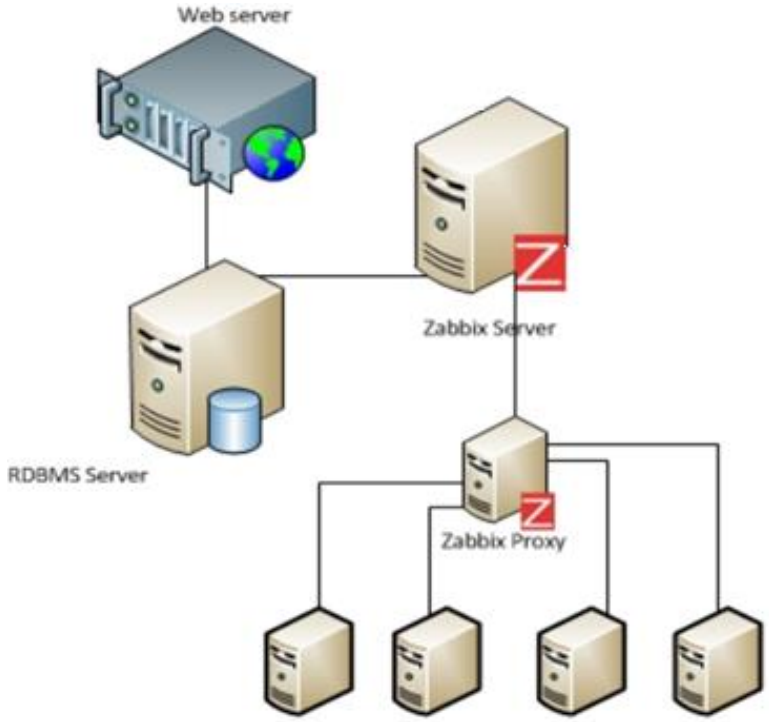
1.6.1. Zabbix architektura

V této podkapitole je vysvětlena architektura monitorovacího nástroje Zabbix. Nejdříve budou představeny možnosti nasazení monitorovacího systému. Monitorovací systém Zabbix lze nasadit a provozovat podle následující topologie (Dalle Vacche, 2015; *Distributed Monitoring*, © 2001-2020):

- Instalace na jeden server
  - Tento princip instalace obsahuje pouze Zabbix server.
  - Tento přístup není vhodný pro monitorování prostředí, které obsahuje velké množství komponent. Nicméně tato instalace je vhodná pro malé prostředí a pro osvojení si základních návyků.
- Instalace na jeden server a více Zabbix proxy
  - Tento princip obsahuje jeden Zabbix server a více Zabbix proxy.
  - Obecně lze říci, že jedna proxy je určena pro monitorování jedné části prostředí.

- Tento princip instalace je zobrazen na následujícím obrázku (Obrázek 5).

- Distribuovaná instalace
  - Tento způsob instalace je dostupný pouze do verze 2.3.



Obrázek 5 – Zabbix architektura

Zdroj: (Dalle Vacche, 2015)

Pro získání lepší představy je architektura představena na výše uvedeném obrázku. Všechny komponenty jsou podrobněji popsány níže:

- Zabbix server – Jedná se o centrální komponentu zodpovědnou za řízení celého monitorovacího procesu. Server je zodpovědný za zpracovávání příchozích dat, definování monitorovacích úloh, definování spouštěčů upozornění a také za samotné odesílání výstražných notifikací. Logicky je rozdělen do tří částí (Dalle Vacche, 2015; *Zabbix Documentation* 4.4, © 2001-2020):
  - Zabbix server – Tato část byla zmíněna výše.
  - Webové rozhraní – Slouží pro jednoduchou práci a konfiguraci monitorovacího systému. Všechny konfigurace jsou posléze uloženy do databáze.

- Databázový server – Zabbix podporuje několik druhů databází. Do databáze jsou ukládána nasbíraná data a všechny konfigurace. Při instalaci je nutné ve všech databázích vytvořit uživatele, který bude Zabbix využívat a přidělit mu požadovaná oprávnění pro čtení a zápis. Podporovanými databázemi jsou (*Zabbix Documentation* 4.4, © 2001-2020):
    - MySQL,
    - Oracle,
    - PostgreSQL,
    - TimescaleDB,
    - IBM DB2,
    - SQLite.

- Zabbix proxy – Jedná se o proces, který shromažďuje příchozí data odeslaná z jednoho nebo více sledovaných zařízení. Data jsou postupně ukládána do paměti cache a až poté přenesena na Zabbix server, kde jsou zpracována a vyhodnocena. Tímto způsobem lze příznivě snížit zátěž a počet potřebných vstupně/výstupních operací na Zabbix serveru. Nutno podotknout, že proxy vyžadují vlastní aplikační a databázový server. Tento způsob topologie je vhodný pro logické členění nebo monitorování vzdálených komponent. (*Zabbix Documentation* 4.4, © 2001-2020)

- Zabbix agent – Aplikace nainstalovaná na prostředí, které má být monitorováno. Hlavním úkolem je sběr informací o zdrojích serveru a běžících aplikacích. Získaná data jsou posléze reportována na Zabbix server nebo na Zabbix proxy, záleží na typu topologie. Pro získávání potřebných informací jsou hojně využívaná systémová volání, což zvyšuje efektivitu a rychlost. Zabbix agent lze nakonfigurovat pro práci ve dvou režimech (*Zabbix Documentation* 4.4, © 2001-2020):
    - Pasivní – Funguje na principu požadavek/odpověď. Agent odpovídá na požadavek na data odeslaný ze Zabbix serveru. Odpověď obsahuje informace o monitorovaných položkách.
    - Aktivní – V případě aktivní konfigurace v předem daných intervalech agent periodicky odesílá informace o monitorovaných položkách. Údaje o tom, které položky mají být monitorovány, jsou získány ze Zabbix serveru před zahájením monitoringu.

- Zabbix Java Gateway – Podpora pro monitorování aplikací přes JMX. Tento démon přijímá instrukce ze Zabbix serveru, následně se pomocí JMX API dotáže aplikace na požadovaná data. Nutno podotknout, že aplikace musí být spuštěna se specifickým konfiguračním parametrem pro JMX (*Zabbix Documentation* 4.4, © 2001-2020):
    - *-Dcom.sun.management.jmxremote* – Umožňuje připojení pomocí JMX z jiného než lokálního zařízení.

V poslední části této podkapitoly jsou shrnuty výhody a nevýhody monitorovacího nástroje Zabbix. (Kovacs, © 2001-2020; *Nagios Vs. Zabbix Vs. PRTG Vs. Spiceworks Vs. Solarwinds Network Performance Monitor*, 2020)

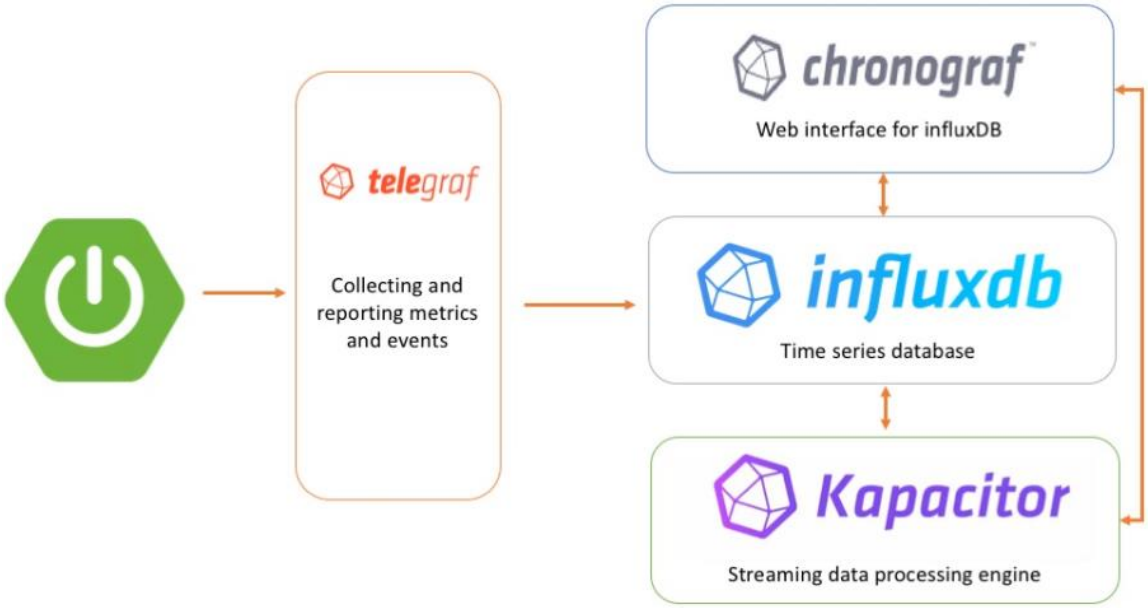
- Výhody:
    - dostupné na Docker Hub,
    - jednoduchá instalace.
- Nevýhody
    - komplikovanější ovládání a nastavení,
    - Zabbix server nelze nainstalovat na operační systém Windows,
    - není vhodný pro instalaci velkého množství zařízení,
    - slabší podpora reportingu.

1.7. TICK Stack

TICK Stack je kolekce open source komponent, které dohromady poskytují služby pro snadné ukládání, vizualizaci časově uspořádaných dat a následné upozorňování. Data jsou získávána pomocí „push“ metody. Tento způsob je velmi vhodný pro monitorování IoT (Internet of Things) komponent a analýzu v reálném čase. Na následujícím obrázku (Obrázek 6) je pro snadnější pochopení a představu předveden jednoduchý nákres topologie všech částí, které jsou zmíněny v následující podkapitole. (*InfluxDB 1.8 documentation*, 2020)

1.7.1. TICK Stack architektura

Zkratka TICK se skládá z následujících komponent: Telegraf, Chronograf, InfluxDB a Kapacitor. Tyto komponenty tvoří celkovou architekturu monitorovacího systému.



Obrázek 6 – TICK architektura

Zdroj: (Aboullaite, 2020)

TICK architektura je složena z následujících komponent, které jsou podrobněji charakterizovány:

Telegraf – Monitorovací agent, který slouží pro získávání metrik ze sledovaných komponent. Telegraf může sbírat metriky z mnoha zařízení (databázové systémy, operační systémy, IoT sensory) a zapisovat do různých výstupních zařízení, takže ho lze spustit na jakémkoliv systému. Je napsán v jazyce GO a to mu umožňuje sběr a výstup dat bez dalších úprav pro určitý typ vstupu nebo výstupu. (*Telegraf*, 2020)

Aktuálně je k dispozici více jak 200 modulů pro zajištění sběru metrik z velkého množství vstupních koncových bodů, kterými mohou být (*Integrations*, 2020): ActiveMQ, Apache Tomcat, Docker, Kafka, Kubernetes a Prometheus. Sesbíraná data mohou být zapsána do velkého množství výstupních zařízení, například: InfluxDB.

- Chronograf – Jedná se o vizualizační nástroj pracující v reálném čase nad daty uloženými v InfluxDB databázi. Chronograf slouží pro vytváření nových pohledů na data a spouštěcích pravidel. Aplikace je naprogramovaná v programovacím jazyce GO. (*Chronograf*, 2020)

- Nástroj Chronograf obsahuje asi dvacet předdefinovaných pohledů na data. Z tohoto vyplývá, že slouží jako prohlížeč pro uložená data v InfluxDB. Vzhledem k tomu je možné vytvářet nové dotazy pro prohlížení dat. Pro dotazování se využívá InfluxQL (Influx Query Language).
- Tento program je velmi podobný vizualizačnímu nástroji Grafana. Hlavní rozdíl mezi komponentami je v počtu možných vizualizačních zdrojů.

- InfluxDB – Databáze určená pro ukládání časově uspořádaných dat. Je naprogramovaná v programovacím jazyce GO a poskytuje velmi rychlý zápis, dostupnost a čtení dat. Komunikace s databází probíhá pouze přes HTTP API. Databáze je vhodná nejen pro ukládání časově uspořádaných dat, ale například i pro ukládání dat, která jsou získána ze senzorů v reálném čase. (*InfluxDB Enterprise*, 2020)
  - Horizontální škálování, vysoká dostupnost, zálohy a podpora databáze jsou dostupné pouze v InfluxDB Enterprise. Avšak databáze poskytuje vysoký výkon i při instalaci pouze na jednom serveru.
- Kapacitor – Nástroj pro zpracovávání dat z InfluxDB naimplementovaný v programovacím jazyce GO. Pomocí tohoto nástroje je možné vytvářet spouštěče upozornění, spouštět ETL (Extract, transform, load) procesy, detekovat nesrovnalosti v získaných datech anebo reagovat na nastalé události například pomocí změny škálování. Kapacitor umožňuje zpracovávat data v reálném čase. (*Kapacitor*, 2020)
  - Lze jej využít pro zprocesování a následnou analýzu dat, před jejich uložením do InfluxDB databáze. Případně jej lze využít k integraci na systém za účelem detekce anomálií nebo programu na porovnávání zdrojů a tím zvýšit jeho účinnost.
  - Upozornění mohou být zasílána pomocí následujících komunikačních kanálů:
    - HipChat,
    - Alerta,
    - Sensu,
    - PagerDuty.

V poslední části této podkapitoly jsou shrnuty výhody a nevýhody monitorovacího nástroje TICK. (Bezalel, 2020)

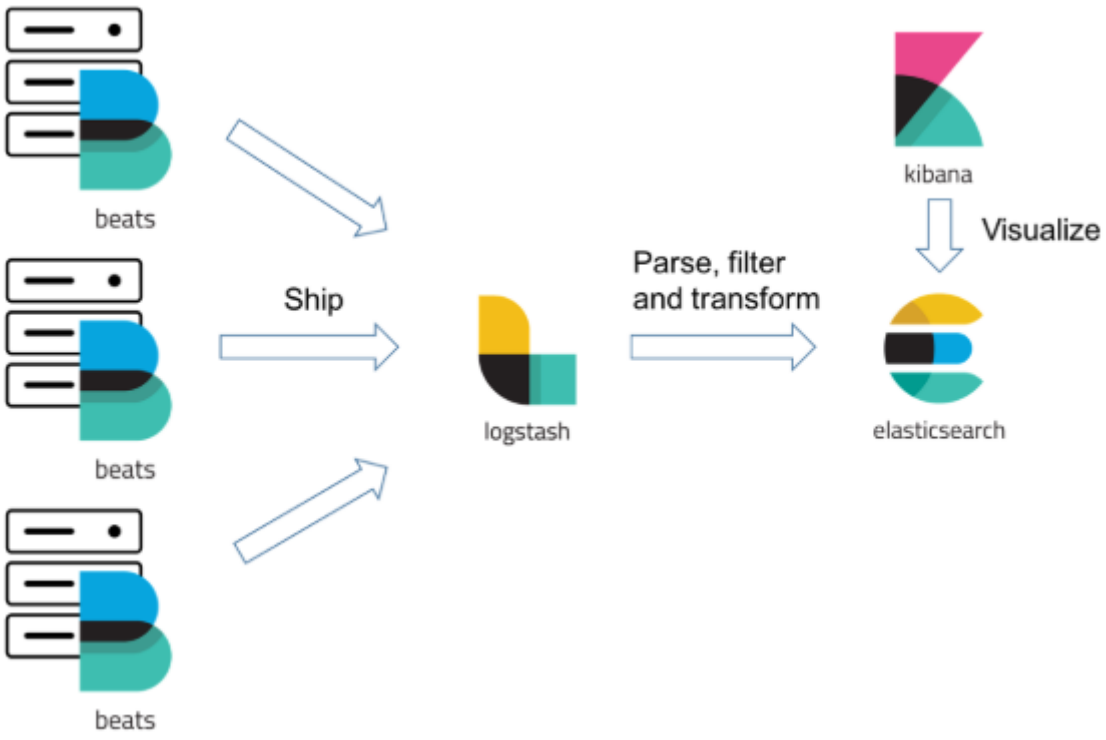
- Výhody:
    - dostupné na Docker Hub,
    - ukládání dlouhodobých dat,
    - jednoduchá instalace.
- Nevýhody:
    - vysoká dostupnost a škálovatelnost je k dispozici pouze v Enterprise verzi,
    - relativně mladý projekt,
    - malá podpora komunity.

1.8. ELK

Jedná se o monitorovací řešení pro zpracovávání a analýzu dat, převážně aplikačních logů. Zkratka ELK se skládá z následujících komponent: Elasticsearch, Logstash, Kibana. Tyto komponenty tvoří dohromady ELK Stack. Zmíněné komponenty jsou spravovány společností Elastic. Kombinace všech komponent slouží pro uložení, zpracování, vyhledání a vizualizaci dat. Hlavním smyslem ELK je vytvoření centralizovaného úložiště, které vede k údržbě všech informací na jednom místě. Nutno podotknout, že všechny zmíněné komponenty jsou open source a jsou dostupné na stránkách Docker Hub. (Jaiswal, 2020)

1.8.1. ELK architektura

Na následujícím obrázku (Obrázek 7) je zobrazena základní architektura ELK. Lze si všimnout, že oproti zmíněným komponentám došlo k rozšíření o komponentu Beats.



Obrázek 7 – ELK architektura

Zdroj: (Seceleanu, 2020)

V této části jsou popsány již zmíněné komponenty podrobněji:

- Beats – Rozšíření nástroje Logstash. Jedná se o odlehčené agenty s cílem sbírat data a jejich následné odesílání. Je možné využívat několika druhů agentů pro sběr z různých částí infrastruktury. (Seceleanu, 2020; *What are Elasticsearch Beats?*, 2020)
    - Filebeat – Načítá data ze souborů. Lze velmi dobře využít pro sbírání aplikačních logů.
    - Metricbeat – Jak z názvu vyplývá, jedná se o agenta, který je využíván pro sbírání metriky ze serverů.
    - Packbeat – Odlehčený agent využívaný pro analýzu síťových protokolů.
    - Winlogbeat – Agent určený pro získávání systémových logů z operačního systému Windows, například přihlašování či instalace nového softwaru.

    - Auditbeat – Stejný princip jako Winlogbeat s tím rozdílem, že je určen pro Unixové operační systémy.
    - Heartbeat – Agent využívaný pro monitoring ostatních komponent pomocí opakovaného dotazování. Dotazy mohou být prováděny pomocí ICMP (Internet Control Message Protocol), TCP (Transmission Control Protocol) a HTTP.

- Logstash – Nástroj určený pro sběr dat z několika zdrojů najednou. Logstash podporuje velké množství možných vstupů, například výše zmíněný Beats. Data jsou následně odeslána do nakonfigurovaného zdroje, nejčastěji do Elasticsearch databáze. Další dostupné cíle pro sběr dat jsou (*LOGSTASH*, 2020):
    - Graphite,
    - JDBC (Java Database Connectivity),
    - Unix.
- Elasticsearch – Analytická, pro vyhledávání určená, NoSQL databáze naprogramovaná v jazyce Java. Aktuálně se řadí mezi jeden z nejpopulárnějších dostupných databázových systémů. Mezi její hlavní domény patří rychlé vyhledávání, analýza a zpracování velkého množství dat v krátkém čase. Vyhledávání je postaveno na open source technologii Apache Lucene. Databáze je velmi dobře horizontálně škálovatelná. V případě nedostatku výkonu, lze přidat nový server do clusteru a tím zvýšit jeho výkon. (Seceleanu, 2020; *ELK Stack Tutorial: Learn Elasticsearch, Logstash, and Kibana*, 2020)
- Kibana – Jedná se o jednoduchý, propracovaný a rychlý vizualizační nástroj. Data pro vizualizaci jsou získávána z Elasticsearch databáze. Umožňuje vytvářet nové pohledy, tabulky, mapy a grafy nad uloženými daty. Data mohou být vykreslována v reálném čase. Využitelné i pro zasílání varovných výstrah. (Seceleanu, 2020; *ELK Stack Tutorial: Learn Elasticsearch, Logstash, and Kibana*, 2020)

V poslední části této podkapitoly jsou shrnuty výhody a nevýhody monitorovacího nástroje ELK. (Jaiswal, 2020)

- Výhody:
    - dostupné na Docker Hub,
    - jednoduchá škálovatelnost,
    - vysoký výkon.
- Nevýhody:
    - chybí vícejazyčná podpora, pouze JSON,
    - instalace a zprovoznění komponent je časově náročné,
    - náročné na systémové zdroje.

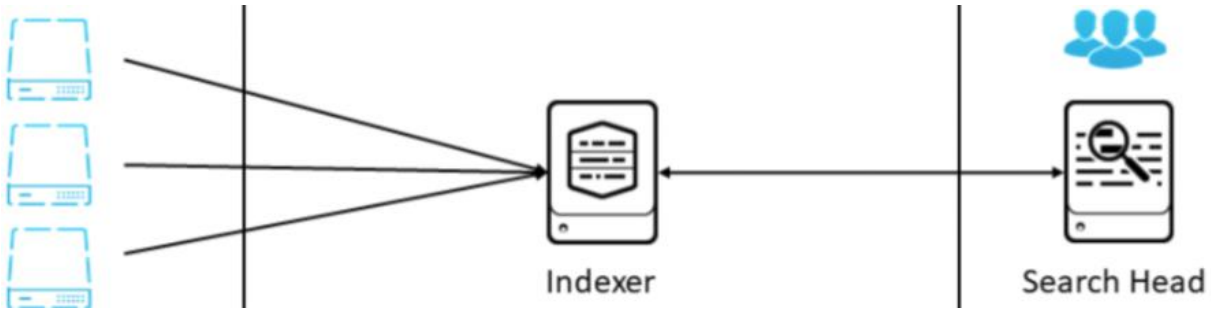
1.9. Splunk

Splunk je technologie, která slouží ke sběru, analýze, monitorování a vizualizaci dat. Získávat data je možné z velkého množství zdrojů, například ze serverů, aplikací, IoT nebo síťových prvků, v různých formátech a v reálném čase. Tato technologie umožňuje přístup k datům z centralizovaného místa. Zpracování dat probíhá ve třech krocích (Vardhan, 2020):

- příjem dat – V této fázi dochází k příjmu prvotních dat, rozdělení dat na bloky a přidání metadat.
- ukládání dat – Skládá se ze dvou hlavních fází:
    - parsování – Získání pouze relevantních dat.
    - indexace – Zápis událostí a indexu na disk. Přítomnost indexu zajišťuje rychlost při následném vyhledávání.
- vyhledávání dat – Kontrola, jak uživatel využívá a prohlíží indexovaná data.

1.9.1. Splunk architektura

Tato podkapitola se zabývá podrobněji architekturou komponent monitorovacího systému Splunk. Princip architektury komponent je zobrazen na obrázku níže (Obrázek 8).



Obrázek 8 – Splunk architektura

Zdroj: (What is Splunk?, 2020)

Architektura Splunk je postavena ze tří základních komponent:

- Forwarder – Komponenta zodpovědná za sběr dat z několika nakonfigurovaných zdrojů. Lze jej rozdělit na dvě další části (*Forwarding Data*, 2020):
    - Universal Forwarder – data jsou zpracovávána pouze minimálně a jsou ihned poslána k Indexeru. Tímto způsobem jsou všechna získaná data odeslána dále a musí být zpracována v další komponentě. Výhodou této komponenty je vyšší úspora hardwarových zdrojů. Nepodporuje uživatelské rozhraní.
    - Light Forwarder – data jsou zpracovávána pouze minimálně a jsou poslána k další komponentě. Nepodporuje tolik funkcí pro předzpracování jako Heavy Forwarder, čímž je snížena spotřeba hardwarových zdrojů a režie.
    - Heavy Forwarder – data jsou předzpracována, rozparsována a indexována v předstihu. Tento postup výrazně usnadňuje následnou práci Indexeru, neboť jsou data již předem připravena. Nevýhodou je vyšší spotřeba hardwarových zdrojů a režie.
- Indexer – Komponenta určená pro indexování a následné ukládání dat. Data jsou uložena do indexů pro urychlení následného vyhledávání. Data jsou zpracovávána dvěma způsoby podle typu přijmutí (*What is Splunk?*, © 2011-2020; *What is Splunk? Beginners Tutorial*, 2020):
    - Universal/Light Forwarder – data nejsou nijak předzpracována. Musí projít analýzou a následujícím uložením.

    - Heavy Forwarder – data jsou již předzpracována a dochází pouze k indexaci dat.
- Search Head – komponenta určená pro vyhledávání a vizualizaci uložených dat. Z vyhledaných dat je možné vytvářet reporty, grafy a dashboardy.

V poslední části této podkapitoly jsou shrnuty výhody a nevýhody monitorovacího nástroje Splunk. (*What is Splunk? Beginners Tutorial*, 2020)

- Výhody:
    - efektivní sběr dat,
    - kvalitně propracované uživatelské rozhraní,
    - free verze pro vývojáře.
- Nevýhody:
    - komerční užití,
    - složitější instalace a náročnější na osvojení,
    - nízká efektivita dotazovacího jazyka.

1.10. Výběr monitorovacího nástroje

Po důkladném zvážení všech kladných i záporných stránek vybraných monitorovacích nástrojů a jejich využitelnosti se pro účely této práce jeví jako nejlepší monitorovací nástroj Prometheus. Jedná se o open source technologii, je tedy dostupná a ekonomicky nenáročná. V úvahu byla vzata jeho jednoduchost používání, vysoká podpora komunity (přínos při budoucím řešení nastalých problémů) a vysoký počet dostupných exportérů. Vzhledem k faktu, že v tomto projektu není plánováno ukládat získaná data po dlouhý časový úsek a není potřeba analyzovat strojová data, odpadly tedy jeho největší nedostatky. Oproti všem ostatním monitorovacím nástrojům je zde pro získávání dat využita metoda „pull“, která je aktuálně netradiční, moderní, a tudíž velmi zajímavá.

2. PRAKTICKÁ ČÁST

V první části této kapitoly je nejdříve detailně popsáno prostředí, které bylo využito pro nasazení, spuštění a následné monitorování pomocí dohledového systému. Podrobně je představena aplikace využitá pro demonstraci monitorovacích úkonů a postup, jak vytvořit aplikaci běžící v kontejneru. Dále jsou popsány základní pojmy i jednotlivé kroky pro správné provedení instalací vybraných dohledových nástrojů, konfigurace monitorovacího systému, nastavení monitorovacích pravidel a vytvoření vlastních metrik. Následně je demonstrována ukázka selhání sledovaného systému a předvedení odesílání varovných notifikací.

2.1. Popis prostředí

Jak již bylo zmíněno výše, celé testovací prostředí je postaveno na principu aplikací běžících v kontejnerech. Využita je kontejnerizace pomocí technologie Docker. Tato aplikace umožňuje vystavět celé testovací, produkční prostředí na jednom zařízení za předpokladu dostatku systémových zdrojů. Připravené testovací prostředí disponuje vysokou mírou přenositelnosti, které zajišťuje pohodlnou migraci mezi prostředími. Je obvyklé, že v praxi je využíváno několik úrovní prostředí, od testovacího prostředí pro vývojáře, uživatelské testovací prostředí, a nakonec produkční prostředí, které je potřeba monitorovat. Možnost vystavět celé prostředí na jednom zařízení zajišťuje vysokou úsporu zdrojů, serverů a potřebných konfigurací, které by bylo potřeba provést.

2.2. Vytvoření Docker kontejneru z vlastní aplikace

Následující kapitoly podrobně popisují aplikaci použitou pro monitorování a všechny další potřebné aplikace pro její správný běh. Následně je definován konfigurační soubor určený pro vytvoření Docker kontejneru, ve kterém je spuštěna naimplementovaná aplikace.

2.2.1. Detailní popis aplikace

Aplikace zvolená pro monitorování je napsána v jazyce Java 1.8 s využitím následujících frameworků:

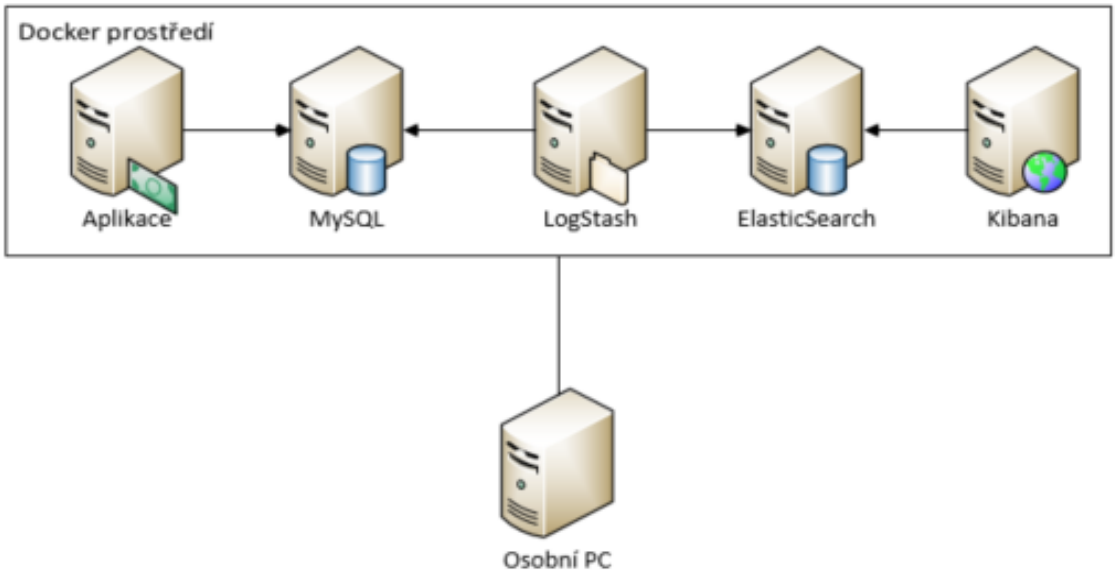
- Spring Boot – Open source framework založený na Spring frameworku určený pro vytváření aplikací připravených pro produkční použití.
- Apache Tomcat – Open source webový server založený na jazyce Java.
- Hibernate – Open source framework využíván pro objektově relační mapování, slouží jako mezivrstva mezi aplikací a databázovým serverem.

- Apache Maven – Framework založený na programovacím jazyce Java, užívaný ke správě balíčků a řízení sestavování programů.
- JUnit – Framework založený na programovacím jazyce Java sloužící k vytváření, spouštění a správu jednotkových testů.
- Thymeleaf – Nástroj pro vytváření šablon pro webové aplikace.

Z pohledu implementace se jedná o Model View Controller (MVC) aplikaci, která slouží pro správu a udržování informací takzvaných „chytrých“ domů a také se zabývá použitými technologiemi v nich. Využívány jsou dvě úrovně oprávnění: administrátor a uživatel. Pro persistenci informací je zvolen databázový server využívající technologii MySQL. K navození produkčních podmínek, kde je často velmi důležité, aby mezi s sebou komunikovalo více aplikací, zde budou využity a nasazeny následující technologie:

- MySQL – Jeden z nejpopulárnějších systémů pro řízení báze dat.
- LogStash – Open source nástroj sloužící pro integraci, získávání, formátování dat, z jednoho nebo více zdrojů a následné uložení dat do jednoho nebo více zdrojů.
    - Přesněji v tomto projektu je využíván pro periodické získávání dat o aktuální teplotě místností z MySQL databáze a jejich ukládání do Elasticsearch databáze.
- Elasticsearch – V současnosti velmi využívaný a velmi oblíbený open source systém pro řízení báze dat. Jedná se o dokumentově orientovaný, fulltextový, analytický nástroj pro ukládání, vyhledávání a správu ukládaných dat. Je zde možné uložit velká množství dat, ke kterým se dá poté snadno přistoupit ve velmi krátkém čase. Další nespornou výhodou je možnost škálovatelnosti a běh v clusteru. Data jsou uložena ve formátu JavaScript Object Notation (JSON) a z výše zmíněného je jasné, že se jedná o NoSQL databázi.
- Kibana – Open source analytický a zobrazovací nástroj, který slouží pro práci s daty uloženými v Elasticsearch databázi. S využitím nástroje Kibana je možné velmi jednoduše vytvářet přehledy dat, grafy, tabulky a mapy a zobrazovat tato data v reálném čase.
    - V tomto projektu slouží pro vizualizace stavů teplot ve všech místnostech chytrých domů.

Celé aplikační prostředí je názorně zobrazeno na následujícím obrázku (Obrázek 9).



Obrázek 9 – Připravené prostředí k monitorování

Zdroj: Vlastní zpracování

2.2.2. Vytvoření kontejneru

Pro vytvoření kontejnerizované aplikace je možné využít dva způsoby:

- pomocí frameworku Maven,
- pomocí souboru Dockerfile.

První možnost je velmi výhodná v případě, kdy je potřeba spravovat Docker images a kontejnery pomocí předefinovaných cílů, například:

- docker:start – vytvoření a start kontejnerů,
- docker:stop – zastavení a odstranění kontejnerů,
- docker:build – vytvoření Docker images,
- docker:push – nahrání Docker images do registrů,
- docker:remove – odstranění Docker images z registrů.

Druhá možnost je mnohem jednodušší a je použita v této práci. Jedná se o vytvoření speciálního souboru Dockerfile v projektu, pomocí kterého dojde k lokálnímu vytvoření Docker image podle nadefinované konfigurace. Konfigurace Dockerfile je zobrazena na obrázku níže (Obrázek 10). (*Dockerfile reference*, 2019)

```
# use base unix image which contains Java runtime
FROM openjdk:8-jdk-alpine

# Add info about maintainer
LABEL maintainer="st47115@student.upce.com"

# set the the path where the volume is pointing at
VOLUME /tmp

# set the port which si exposed outside of the container
EXPOSE 8080

# set the application's jar file
ARG JAR_FILE=target/sh_application-0.0.3.jar

# Add the application's jar to the container
ADD ${JAR_FILE} sh_application.jar

# Run the jar file during the start
ENTRYPOINT ["java","-Djava.security.egd=file:/dev/./urandom","-jar","/sh_application.jar"]
```

Obrázek 10 – Dockerfile konfigurace

Zdroj: Vlastní zpracování

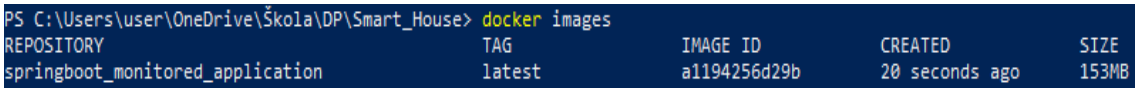
Jednotlivé instrukce jsou charakterizovány následovně:

- FROM – Slouží pro výběr Docker Image, která bude sloužit jako základ nového kontejneru. V tomto případě se jedná o Alpine Linux s již nainstalovaným OpenJDK 1.8. Alpine Linux je pro použití v Docker kontejnerech vhodný vzhledem k jeho velmi malé celkové velikosti. Tím nám umožňuje udržovat výsledný kontejner v nejmenší možné velikosti.
- LABEL – Umožňuje přidání metadat do Docker image.
- VOLUME – Vytvoří přístupový bod, do kterého je možné vstupovat zvenčí z hostitelského prostředí. Zjednodušeně se jedná o místo, kam mohou být nakopírovány soubory při vytváření kontejneru.
- EXPOSE – Je využíván k vystavení požadovaného portu, na kterém kontejner naslouchá.
- ARG – Slouží k ukládání proměnných, které je možné později využít.
- ADD – Užívá se ke kopírování souborů nebo složek do souborového systému kontejneru.
- ENTRYPOINT – Definice příkazů, které se používají při startu kontejneru. V tomto projektu, bude spuštěn soubor sh_application.jar.

Po správném nakonfigurování se po spuštění následujícího příkazu vytvoří Docker image, ze kterého je možné vytvářet další kontejnery (instance). Nutno podotknout, že pro spuštění následujícího příkazu, je nutné být ve stejném adresáři se souborem Dockerfile.

docker build -t springboot_monitored_application .

Pokud všechny operace proběhly v pořádku, měla by po zadání příkazu Docker images být vidět nově vytvořená Docker image s názvem: *springboot_monitored_application* obdobně jako na následujícím obrázku (Obrázek 11).



Obrázek 11 – Docker image

Zdroj: Vlastní zpracování

Nyní je aplikace připravena pro nasazení a nastartování. Tato procedura je předvedena v následujících kapitolách.

2.3. Instalace monitorovacího nástroje Prometheus

Prometheus je open source nástroj pro monitoring a následné odesílání informačních notifikací v případě problému. Jak bylo zmíněno v teoretické části, sběr metrik probíhá periodicky pomocí nastaveného intervalu. Metrika je sbírána z HTTP bodů cíle, který byl označen pro sledování.

Obdobně jako v předchozích případech se využívá instalace pomocí technologie Docker. Monitorovací nástroj Prometheus je dostupný na online úložišti pro Docker images, které se nazývá: Docker Hub. Pro stažení image na lokální úložiště je potřeba zadat do příkazové řádky následují příkaz:

docker pull prom/prometheus

Po stáhnutí image je dalším nutným krokem správné nastavení pro úspěšné nastartování aplikace v kontejneru. Pro získání co největší míry přenositelnosti využijeme konfigurační nástroj Docker-Compose. Tento nástroj slouží pro správu a management většího množství různorodých kontejnerů prostřednictvím jednoho konfiguračního souboru. Možnost centralizované konfigurace výrazně zvyšuje přehlednost a přenositelnost celého prostředí, protože s pouze několika málo úpravami, například cesty k souborům, je možné vytvořit nové prostředí na úplně jiném zařízení. Konfigurační soubor je psán v jazyce YAML Ain't Markup

Language (YAML). V následující části bude předvedena a detailně popsána konfigurace pro spuštění monitorovacího nástroje Prometheus. (*Overview of Docker Compose*, 2019)

2.3.1. Spuštění nástroje Prometheus pomocí Docker Compose

Na obrázku níže (Obrázek 12) je detailně popsána konfigurace ke správnému vytvoření kontejneru s nástrojem Prometheus promocí Docker-Compose.

```
# Prometheus configuration
  prometheus:
    image: prom/prometheus:latest
    container_name: dockerenvironment_prometheus
    ports:
      - 9090:9090
    volumes:
      - "./prometheus/prometheus.yml:/etc/prometheus/prometheus.yml"
      - "./prometheus/prometheus_db:/var/lib/prometheus"
    command:
      - --config.file=/etc/prometheus/prometheus.yml
    networks:
      dt-network:
        ipv4_address: "172.32.10.7"
```

Obrázek 12 – Prometheus – Docker Compose

Zdroj: Vlastní zpracování

Význam jednotlivých parametrů je následující:

- image – Definuje, ze kterého Docker image bude kontejner vystavěn a jaká verze bude použita.
    - prom/prometheus – Image dostupný na stránkách Docker Hub.
    - latest – Udává, že se využívá poslední dostupná verze 2.15.
- container_name – Vyjadřuje výsledný název kontejneru, případně se použije náhodně vygenerovaný název.
- ports – Uvádí, na jakém portu bude Prometheus dostupný.
    - 9090:9090 – přesměrování portu kontejneru z 9090 na lokální počítač – 9090
- volumes – Slouží k vytvoření přístupového bodu v kontejneru.
    - V kontejneru je konfigurační soubor dostupný v adresáři: /etc/prometheus/.
- command – Je využíván ke spouštění specifických příkazů při startu kontejneru.

- parametr --config.file – Udává cestu ke konfiguračnímu souboru pro nástroj Prometheus.

- networks – Určuje, v jaké síti se bude kontejner nacházet.
  - dt-network – Název sítě vytvořené pro tento projekt.
    - ivp4_address – IP adresa přidělená kontejneru.

Takto vytvořená konfigurace umožní spustit nástroj Prometheus, který pracuje na přidělené IP adrese s portem 9090.

2.3.2. Konfigurace nástroje Prometheus

Jak bylo zmíněno v kapitole výše, konfigurační soubor je alokován mimo Docker kontejner a je možné ji upravovat libovolně bez nutnosti přístupu do kontejneru. Konfigurační soubor je uložen v YAML formátu, takže je důležité, aby při konfiguraci nedocházelo ke špatnému formátování textu, což může vést k problémům při startu, nebo později k nesprávnému a neočekávánému chování. Využitá konfigurace je zobrazena na obrázku č. 13, následně detailně popsána v odstavci situovaném pod obrázkem (*CONFIGURATION*, © 2014-2020).

```
#  configuration part used for obtaining the metrics from the containers
scrape_configs:
  - job_name: 'prometheus'
    scrape_interval: 10s
    scrape_timeout: 5s
    metrics_path: '/metrics'
    static_configs:
      - targets: ['172.32.10.7:9090']
  - job_name: 'node-exporter'
    static_configs:
      - targets: ['172.32.10.8:9100']
  - job_name: 'docker'
    static_configs:
      - targets: ['10.42.10.132:9323']
  - job_name: 'cadvisor'
    static_configs:
      - targets: ['172.32.10.9:8080']
```

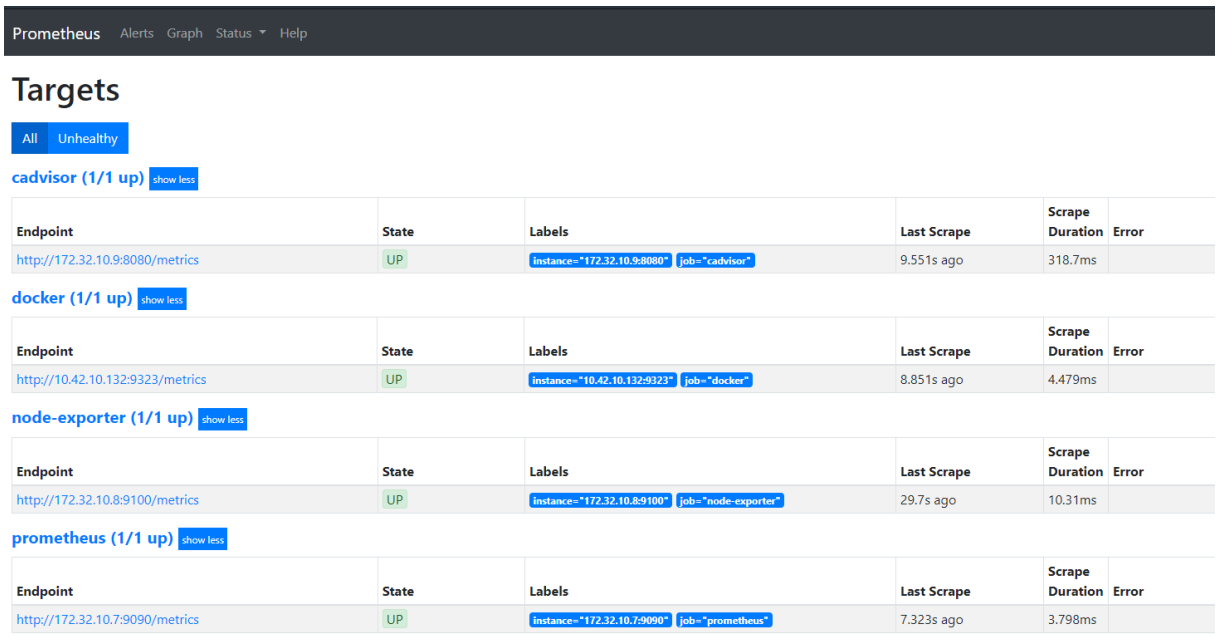
Obrázek 13 – Konfigurace nástroje Prometheus

Zdroj: Vlastní zpracování

Význam jednotlivých parametrů je následující:

- global – V konfigurační sekci global jsou definovány parametry, které jsou platné pro všechny ostatní sub části konfigurace. Všechny nadefinované parametry je poté možné překrýt specifičtější konfigurací.
    - scrape_interval – Uvádí, jak často dochází ke sbírání dat z monitorovaných zařízení.
    - scrape_timeout – Definuje, za jakou dobu vyprší požadavek na sběr dat z monitorovaného zařízení.
    - evaluation_interval – Vymezuje, jak často dochází k porovnávání nastavených monitorovacích pravidel.
- scrape_configs – Formuluje určitou sadu cílů, ze kterých se získává metrika pro monitoring. Cíle, které jsou určené pro monitoring, je možné nastavit dynamicky anebo staticky. V tomto projektu se využívá statické nastavení.
    - job_name – Název cíle, který je nutné monitorovat.
    - scrape_interval – Uvádí, jak často dochází ke sbírání metriky. Tato konfigurace má přednost před využitím globální konfigurace. Pokud není vyplněna, využije se hodnota z globální konfigurace.
    - scrape_timeout – Definuje, za jakou dobu vyprší požadavek na sběr dat z monitorovaného zařízení. Tato konfigurace má přednost před využitím globální konfigurace. Pokud není vyplněna, využije se hodnota z globální konfigurace.
    - metrics_path – HTTP cesta, ze které se stáhne metrika. Výchozí hodnota tohoto parametru je: */metrics.*
    - static_configs – Jedná se o statické nastavení cíle, který má být monitorován. V závislosti na typu úlohy je možné definovat i více cílů jako list.
        - targets – Představuje statický list cílů. Využívá se IP adresa a číslo portu, na kterém je služba vystavena.

Takto nakonfigurovaný nástroj Prometheus připravený pro sběr metriky a monitorování. Proces nastavení upozorňovacích pravidel je předveden v následujících kapitolách. Jak je zřejmé z uvedené konfigurace, v tomto projektu budou monitorovány celkem 4 cíle, což je zobrazeno na následujícím obrázku (Obrázek 14).



Obrázek 14 – Nakonfigurované cíle pro sběr metriky

Zdroj: Vlastní zpracování

Technologie využité pro získání a vystavování metriky jsou podrobně popsány v následujících kapitolách.

2.4. Popis a konfigurace podpůrných nástrojů pro monitoring

V následujících podkapitolách jsou rozebrány nástroje využitelné ke sběru metrik pro nástroj Prometheus.

2.4.1. Node exporter

Nástroj Node exporter slouží pro získávání metriky z hostitelského zařízení, na kterém je spuštěno celé prostředí. Tento nástroj vystavuje uvedenou metriku, aby mohla být následně zpracována Prometheem. Umožňuje získávat informace o využití paměti, souborového systému, CPU, běžících, respektive o blokovaných procesech a následně i jejich agregované statistiky. Nástroj je napsán v programovacím jazyce GO a obsahuje velké množství sběračů metriky, které je možné aktivovat či deaktivovat pomocí konfiguračního souboru (*Node exporter*, 2020).

Na následujícím obrázku (Obrázek 15), je zobrazen výpis konfigurace obsahující parametry umožňující spuštění nástroje Node exporter pomocí Docker-Compose.

```yaml
# Node exporter configuration
  node-exporter:
    image: prom/node-exporter
    container_name: dockerenvironment_node-exporter
    ports:
      - 9100:9100
    links:
      - prometheus
    depends_on:
      - prometheus
    environment:
      - JAVA_OPTS=-Xms64M -Xmx256M
    networks:
      dt-network:
        ipv4_address: "172.32.10.8"
```

Obrázek 15 – Node exporter – Docker Compose

Zdroj: Vlastní zpracování

Vzhledem k popisu jednotlivých částí Docker-Compose v jedné z předchozích kapitol jsou zde popsány pouze části, kde došlo ke změně:

- image
    - prom/node-exporter – Image dostupný na stránkách Docker Hub.
    - latest – Uvádí, že se využívá poslední dostupná verze 0.18.1.
- depends_on – Vyjadřuje závislost mezi kontejnery.
    - prometheus – Kontejner node-exporter, bude nastartován až po nastartování kontejneru Prometheus.

Pro zařízení, která využívají operační systém typu Windows, se využívá nástroj: WMI exporter. Několik nejzajímavějších sběračů je popsáno v tabulce níže:

Tabulka 1 – Node exporter – Sběrače informací/statistik

Název	**Detail**	**Výchozí stav (Aktivováno/Deaktivováno)**
cpu	Získávání informací z centrální procesorové jednotky	Aktivováno
filesystem	Získávání informací/statistik ze souborového systému	Aktivováno
interrupts	Získávání statistiky ohledně přerušení operačního systému	Deaktivováno
loadavg	Získávání informací o zatížení operačního systému	Aktivováno
meminfo	Získávání informací/statistik o využití paměti na hostitelském zařízení	Aktivováno
processes	Získávání informací statistik o procesech – data jsou získávána z adresáře /proc/	Deaktivováno

Zdroj: Vlastní zpracování na základě dat z GitHub (Node exporter, 2020)

Všechny metriky, které jsou získány, jsou následně vystaveny na následující cestě: *IP:PORT/metrics*. V tomto projektu bude metrika vystavena na portu: 9100. Některé zajímavé vystavené metriky jsou detailněji popsány níže:

- node_boot_time_seconds – Specifikuje čas spuštění hostitelského zařízení.
- node_load – Stanovuje průměrnou hodnotu zátěže CPU.
    - node_load1 – za 1 minutu
    - node_load5 – za 5 minut
    - node_load15 – za 15 minut

- node_filesystem_size_bytes – Celková velikost souborového systému, hodnota je udávána v bytech.
- node_filesystem_avail_bytes – Dostupná velikost souborového systému pro uživatele, kteří nejsou *root*, hodnota je udávána v bytech.
- node_procs_blocked – Počet procesů, které jsou v blokovaném stavu a čekají na dokončení vstupně výstupní operace.
- node_disk_reads_completed_total – Celkový počet úspěšně provedených čtení z disku.
- node_disk_writes_completed_total – Celkový počet úspěšně provedených zápisů na disk.

2.4.2. Cadvisor

Monitorování hostitelského zařízení je zajištěno pomocí nástroje Node exporter. V této kapitole je předvedeno nakonfigurování a seznam základních metrik pomocí nástroje Cadvisor. Tento nástroj se používá pro získávání metriky všech běžících kontejnerů a následně dojde k vystavení této metriky, aby mohla být zpracována Prometheem. Nástroj Cadvisor primárně slouží pro získávání informací o kontejnerech, například o jejich využití paměti, CPU, souborovém systému či síťovém provozu (*Cadvisor*, 2020).

Na uvedeném obrázku (Obrázek 16) je zobrazena konfigurace potřebná pro spuštění nástroje Cadvisor pomocí Docker-Compose:

```
#cadvisor exporter configuration
 cadvisor:
   image: google/cadvisor:latest
   container_name: dockerenvironment_cadvisor
   ports:
     - '10090:8080'
   volumes:
     - /:/rootfs:ro
     - /var/run:/var/run:rw
     - /sys/fs/cgroup/cpu:/sys/fs/cgroup/cpu
     - /sys/fs/cgroup/cpuacct:/sys/fs/cgroup/cpuacct
     - /var/lib/docker/:/var/lib/docker:ro
     - /cgroup:/cgroup:ro
   command:
     - --enable_load_reader=true
   depends_on:
     - prometheus
   networks:
     dt-network:
       ipv4_address: "172.32.10.9"
```

Obrázek 16 – Cadvisor – Docker Compose

Zdroj: Vlastní zpracování

Zde jsou popsány části nutné k pochopení konfigurace z obrázku č. 16.

- image
    - google/cadvisor – Image dostupný na stránkách Docker Hub.
    - latest – Udává, že bude využita poslední dostupná verze 0.32.0.

Z výše uvedeného obrázku je zřejmé, že nástroj Cadvisor potřebuje mít připojená některá úložiště z hostitelského zařízení, ze kterých jsou získávány informace a statistiky.

Některé zajímavé metriky získané pomocí nástroje Cadvisor jsou popsány níže:

- cadvisor_version_info – Slouží k vypsání základních informací o nástroji Cadvisor.
- container_cpu_load_average_10s – Formuluje průměrnou hodnotu zatížení kontejneru za posledních 10 s.
- container_cpu_system_seconds_total – Celkový čas spotřebovaný CPU.
- container_fs_inodes_free – Definuje počet dostupných i-uzlů.

- container_fs_limit_bytes – Specifikuje počet bytů hostitelského zařízení, které mohou být využity kontejnerem.
- container_last_seen – Jedná se o časové razítko, kdy byl kontejner naposledy zpozorován.
- container_tasks_state – Udává počet kontejnerů podle jednotlivých stavů (spící, běžící, zastavený, nepřerušitelný spánek).
- container_network_receive_bytes_total – Definuje celkový počet přijatých bytů.
- container_memory_swap – Stanovuje využití paměti swap kontejnerem.

Nástroj Cadvisor všechny získané metriky vystaví na následující cestě: *IP:PORT/metrics*. V tomto projektu bude metrika vystavena na portu 10090.

2.4.3. Docker deamon metrics

Stejně jako příklady zmíněné výše, Docker podporuje monitoring pomocí nástroje Prometheus. Pomocí jednoduché konfigurace je možné vystavit metriku pro monitoring Docker kontejnerů a Docker démona. Pro vystavení metriky na operačním systému Linux je nutné upravit konfiguraci. Konfigurace je uložena v následující cestě: /etc/docker/daemon.json. Obsah konfigurace by měl vypadat následovně (*Collect Docker metrics with Prometheus*, 2019):

```
{
 "metrics-addr": "IP:9323",
 "experimental": true
}
```

Po provedené změně konfigurace je zapotřebí restartovat Docker, aby došlo k propsání změn. Po restartu bude metrika vystavena na adrese: IP:9323*/metrics*. Některé vybrané vystavené metriky jsou popsány níže (*Collect Docker metrics with Prometheus,* 2019):

- process_start_time_seconds – Specifikuje čas, kdy došlo ke startu docker procesu.
- engine_daemon_container_states_containers – Uvádí počet kontejnerů podle stavu (pozastavený, zastavený, běžící).
- engine_daemon_health_checks_failed_total – Definuje počet neúspěšných pokusů o zjištění stavu kontejneru.

Po úspěšném nakonfigurování a instalaci výše zmíněných nástrojů, by měl stav na nástroji Prometheus reflektovat s obsahem uvedeným na obrázku č. 14. Pokud konfigurace proběhla úspěšně, měla by být pro nástroj Prometheus dostupná většina metrik pro monitorování Docker kontejnerů a hostitelského zařízení, které byly zmíněny v předešlých kapitolách.

2.4.4. Spring Boot Actuator

Všechny metriky, zmíněné v kapitolách výše, sloužily spíše pro získávání informací o aktuálně běžících kontejnerech včetně hardwarových statistik. Pro získání informací a statistik přímo z běžící aplikace je možné využít komponentu Spring Boot Actuator. Tato technologie je součástí již běžící aplikace a obsahuje již hotové části aplikace, prostřednictvím kterých lze monitoring uskutečnit. Technologii lze do aplikace přidat, za předpokladu, že je pro správu závislostí využit nástroj Maven, pomocí následující části kódu (*Spring Boot Actuator*, 2020; *Spring Boot Actuator: Production-ready Features*, 2020):

```
<dependency>
    <groupId>org.springframework.boot</groupId>
    <artifactId>spring-boot-starter-actuator</artifactId>
</dependency>
```

Po přidání závislosti do programu jsou k dispozici následující koncové body (endpointy) pro monitoring:

- /auditevents – Vystavuje list informací o auditních operacích.
- /beans – Poskytuje kompletní list všech dostupných beans využitých v aplikaci.
- /caches – Zpřístupní dostupné cache.
- /env – Vrací aktuální konfigurace prostředí.
- /flyway – Poskytne informace o stavu nástroje Flyway, který slouží pro udržování konzistentního stavu databáze.
- /health – Vrací stav aplikace.
- /heapdump – Dojde k vytvoření a poskytnutí JVM otisku aplikace.
- /info – Základní informace o aplikaci (číslo buildu, poslední commit).
- /liquibase – Poskytne informace o stavu nástroje Liquibase, který slouží pro udržování konzistentního stavu databáze.

- /logfile – Poskytne obyčejné logy z aplikace – stejné, které jsou zapisovány do logovacího souboru.
- /loggers – Umožní změnit úroveň logovacích nástrojů v aplikaci – například z INFO na DEBUG.
- /metrics – Zobrazí detailní metriku s informacemi o aplikaci.
- /prometheus – Poskytne metriku, která je obsahově stejná jako předchozí, ale je zformátovaná, aby mohla být získána pomocí monitorovacího nástroje Prometheus.
- /scheduledtasks – Poskytne detailní seznam všech naplánovaných úkolů v aplikaci.
- /sessions – Seznam HTTP relací, které Spring Boot využívá.
- /shutdown – Provede se vypnutí aplikace – v základu je tato možnost vypnutá.
- /threaddump – Po přístupu na tento koncový bod dojde k vytvoření otisku aktuálně běžících vláken v aplikaci.

Vzhledem k rozsahu všech možných činností, výše zmíněných endpointů, není vhodné je ponechat nezabezpečené a konfigurovatelné (vystavené ven do světa). Pro potřeby tohoto projektu je k dispozici pouze koncový bod */prometheus*. Tohoto stavu dosáhneme níže uvedeným konfiguračním parametrem (*Spring Boot Actuator*, 2020; *Spring Boot Actuator: Production-ready Features*, 2020).

management.endpoints.web.exposure.include=prometheus

Pro získání přístupu k nově vystavené metrice, je nutné zadat do webového prohlížeče následující adresu: *IP:PORT/actuator/prometheus*. V případě správné konfigurace získáme obdobný výstup jako je uvedeno na obrázku níže (Obrázek 17).

```
# HELP tomcat_threads_busy_threads
# TYPE tomcat_threads_busy_threads gauge
tomcat_threads_busy_threads{name="http-nio-8080",} 1.0
# HELP jdbc_connections_active
# TYPE jdbc_connections_active gauge
jdbc_connections_active{name="dataSource",} 0.0
# HELP jvm_threads_states_threads The current number of threads having NEW state
# TYPE jvm_threads_states_threads gauge
jvm_threads_states_threads{state="runnable",} 6.0
jvm_threads_states_threads{state="blocked",} 0.0
jvm_threads_states_threads{state="waiting",} 12.0
jvm_threads_states_threads{state="timed-waiting",} 5.0
jvm_threads_states_threads{state="new",} 0.0
jvm_threads_states_threads{state="terminated",} 0.0
```

Obrázek 17 – Spring Boot Actuator metrika

Zdroj: Vlastní zpracování

Nyní je dalším důležitým krokem nastavit nový cíl, který se použije pro monitoring pomocí nástroje Prometheus. Pro přístup k jakékoliv části vytvořené aplikace je vyžadována autorizace uživatele. Toto je důvodem k nastavení autorizace i na nástroj Prometheus. Potřebná konfigurace je zobrazena na následujícím obrázku (Obrázek 18).

Mimo základní autorizace, myšleno uživatelské jméno a heslo, je možné využít i další typy autorizace. Například pomocí tokenů. Tato vlastnost v tomto projektu nebude předvedena.

```
- job_name: 'springboot_app'
  metrics_path: 'actuator/prometheus'
  basic_auth:
    username: 'prometheus'
    password: 'prometheus'
  static_configs:
    - targets: ['172.32.10.6:8080']
```

Obrázek 18 – Konfigurace nástroje Prometheus s autorizací

Zdroj: Vlastní zpracování

Po úspěšném nakonfigurování jsou k dispozici nové metriky.

- logback_events_total – Udává počet zalogovaných událostí roztříděných podle úrovně (info, debug, error, warn, trace).
- jvm_threads_states_threads – Specifikuje počet vláken podle stavu (runnable, blocked, waiting, new, terminated).
- process_files_open_files – Vyjadřuje aktuální počet aplikací otevřených souborů.
- process_cpu_usage – Stanovuje aktuální využití CPU pro proces JVM.

Všechny výše zmíněné metriky jsou nadefinovány komponentou Actuator a mnohdy nepokryjí požadavky, které jsou na monitorování kladeny ze strany projektu nebo zákazníka. Pod tím je možné představit si metriky, které přímo souvisí s funkčností aplikace. Například: datum vypršení certifikátů určených pro komunikaci, počet nezpracovaných úloh nebo zpráv či doba probíhajícího výpočtu. Jednou z možností, jak vytvořit vlastní metriky, je využití komponenty Micrometer.

2.4.5. Micrometer

Komponenta Micrometer dává k dispozici jednoduché aplikační rozhraní, které je vsazené přímo do implementace a které lze využít pro nejmodernější monitorovací systémy. V této podkapitole bude předvedeno přidání komponenty Micrometer do aplikace a následně

vytvoření vlastní metriky pro monitorovací nástroj Prometheus. Pro začátek je nutné přidat závislost do aplikace pomocí nástroje Maven. Potřebná část kódu je zobrazena níže (*Micrometer Prometheus*, 2017):

```
<dependency>
    <groupId>io.micrometer</groupId>
    <artifactId>micrometer-registry-prometheus</artifactId>
</dependency>
```

Po přidání závislosti je možné přejít k další části, kde bude předvedena ukázková implementace definice vlastní metriky. Metrika, kterou lze vytvořit se dělí na několik částí.

Count

První z nich je metrika založená na principu obyčejného čítače. Čítač, lze pouze zvyšovat a pokus o snížení jeho hodnoty bude neúspěšný a hodnota zůstane nezměněna. Níže je předvedena implementace čítače, který slouží pro získávání počtu neúspěšných přístupů do běžící aplikace. Monitorováním tohoto parametru se výrazně zvyšuje šance na odhalení možného útoku pomocí slovníkové metody prostřednictvím, které by útočník mohl získat přístup do aplikace. (*Quick Guide to Micrometer*, 2020)

```
public UserDetails loadUserByUsername(String username) {
    User user = getRegisteredUsers(username);
    if (user == null) {
        logger.warn( message: "User: {} is not registered, can not be logged in!", username);
        refusedToLogin.increment();
        throw new UsernameNotFoundException(username);
    } else {
        logger.info( message: "User: {} has been logged in successfully!", username);
        successfullyLogged.increment();
        return new org.springframework.security.core.userdetails.User(
                user.getUsername(), user.getPassword(), getAuthorities(user));
    }
}

private void initMonitoringMetrics(MeterRegistry meterRegistry) {
    successfullyLogged = Counter.builder( name: "app.log.requests")
            .tag("status", "successful")
            .description("The number of log requests")
            .register(meterRegistry);
    refusedToLogin = Counter.builder( name: "app.log.requests")
            .tag("status", "unsuccessful")
            .register(meterRegistry);
}
```

Obrázek 19 – Micrometer – Využití komponenty Counter

Zdroj: Vlastní zpracování

Jak z výše uvedeného obrázku (Obrázek 19) vyplývá, jsou využity dvě komponenty typu Counter, které budou zvyšovány v závislosti na počtu úspěšných, respektive neúspěšných přístupů. Volání metody increment zvyšuje čítač v závislosti na typu přístupu, zda byl pokus o přihlášení úspěšný nebo neúspěšný. Pro dodržení osvědčených postupů je důležité, aby název nové metriky dodržoval požadovanou jmennou konvenci. Tagy slouží pro odlišení stejných metrik, které mají odlišný význam. V tomto případě se jedná o odlišení úspěšných a neúspěšných počtů přihlášení.

Gauge

Tento typ metriky znamená v předkladu měřidlo. Od čítače se odlišuje ve více bodech. První bod je absence metod, které jsou použité pro změny stavu, jako tomu bylo u čítače. Měřič vždy získává aktuální hodnotu sledovaného parametru a tato hodnota bude vystavená pro monitorování. Jinak by se dalo říci, že změna hodnot na měřiči nastane tehdy, když dojde k pozorování. Příklad využití měřiče je předveden na obrázku níže (Obrázek 20), kde dochází k měření nejvyšší hodnoty počtu vláken pro JVM (*Quick Guide to Micrometer*, 2020):

```
private void initializeGauge(MeterRegistry meterRegistry) {
    Gauge.builder( name: "app.threads.top", threadBean, ThreadMXBean::getPeakThreadCount)
            .description("Thread the peak live thread count for JVM")
            .register(meterRegistry);
}
```

Obrázek 20 – Micrometer – Využití komponenty Gauge

Zdroj: Vlastní zpracování

Timer

Časovače slouží k měření krátkodobých operací a událostí. Časovač bude vždy hlásit alespoň celkový čas trvání operace. Můžeme například zaznamenávat událost aplikace, která může trvat několik desítek vteřin, ale zároveň nesmí překročit minutu. Na následujícím obrázku (Obrázek 21) je předvedeno využití časovače s použitím anotace Timed, která označí metodu, která bude sloužit pro měření. (*Quick Guide to Micrometer*, 2020)

```java
@Timed(value = "app.scheduled.job")
@Scheduled(cron = "1 * * * * ?")
public void scheduleTaskWithCronExpression() {
    logger.info("Temperature changer job has started!");
    int roomsCount = temperatureManager.changeTemperatureForAllRooms();
    logger.info( message: "Temperature changer job has finished." +
            " Temperatures has been changed for: {} rooms!", roomsCount);
    temperatureManager.calculateAverageHouseTemperature();
}
```

Obrázek 21 – Micrometer – Využití komponenty Timer

Zdroj: Vlastní zpracování

Všechny tímto způsobem vytvořené metriky jsou dostupné pro monitoring na adrese koncového bodu: */actuator/prometheus*. Po provedení všech kroků zmíněných v předchozích kapitolách jsou aktuálně připraveny všechny nástroje, které jsou potřeba pro sběr metrik z celého prostředí.

Pro vytváření monitorovacích a upozorňovacích pravidel je možné využít monitorovací nástroj Prometheus, který aktuálně slouží pro sběr metrik. Vytváření monitorovacích a varovných pravidel je zpočátku časově náročnější. V tento okamžik se hlásí o slovo nástroj Grafana, který dokáže značně zkrátit čas potřebný pro naučení a pochopení základních principů.

2.5. Grafana

Nástroj Grafana je open source nástroj, který je využíván pro vizualizaci a vytváření dashboardů nad již získanými daty. Grafana sama o sobě neukládá nashromážděná data, ale pouze čerpá data z nakonfigurovaného zdroje, nad nimiž je poté možné provádět vizualizace, analýzy a upozorňování. Velkou výhodou je výrazná databázová nezávislost, která umožňuje poměrně velký rozsah používaných technologií. Grafické provedení je na velmi vysoké úrovni, je snadno ovladatelné a intuitivní. Další nespornou výhodou je možnost exportu již vytvořených dashboardů a vizualizací a jejich následné sdílení mezi projekty, za předpokladu využití stejných typů metrik. (*Grafana*, 2020)

V následujících kapitolách je představeno základní nakonfigurování vizualizačního nástroje Grafana a propojení s monitorovacím nástrojem Prometheus. Jak vyplývá z předchozích kapitol, všechny nástroje jsou spuštěny pomocí technologie Docker. Na následujícím obrázku (Obrázek 22) je zobrazena konfigurace potřebná pro spuštění nástroje Grafana pomocí Docker-Compose:

```
# grafana configuration
  grafana:
    image: grafana/grafana:latest
    container_name: dockerenvironment_grafana
    ports:
      - '3000:3000'
    links:
      - prometheus:prometheus
    volumes:
      - ./grafana/data:/var/lib/grafana
      - ./grafana/grafana.ini:/etc/grafana/grafana.ini
    networks:
      dt-network:
        ipv4_address: "172.32.10.10"
```

Obrázek 22 – Grafana – Docker Compose

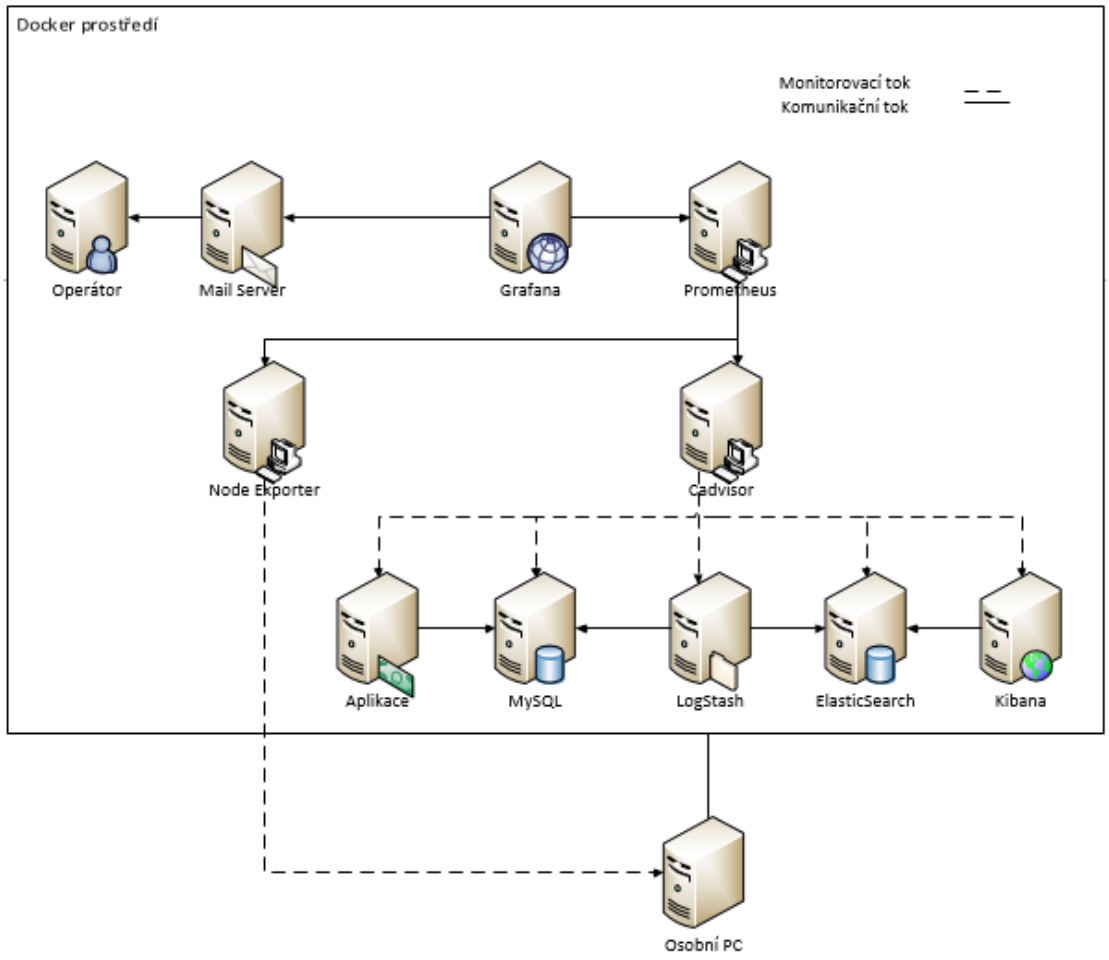
Zdroj: Vlastní zpracování

Zde jsou popsány části nutné k pochopení konfigurace z obrázku č. 22.

- image
    - grafana/grafana – Image dostupný na stránkách Docker Hub.
    - latest – Udává, že bude využita poslední dostupná verze 6.6.1.

Po spuštění bude aplikace dostupná na adrese IP:3000. Základní přihlašovací údaje jsou: admin/admin. Při prvním přihlášení bude zobrazena výzva pro změnu hesla. V zájmu bezpečnosti je doporučeno heslo změnit.

V tuto chvíli jsou všechny potřebné nástroje pro sbírání metrik, monitoring a posílání upozorňovacích notifikací nastavené. Na následujícím obrázku (Obrázek 23) je pro větší představivost zobrazena celá aplikační struktura:



Obrázek 23 – Grafické znázornění celého prostředí

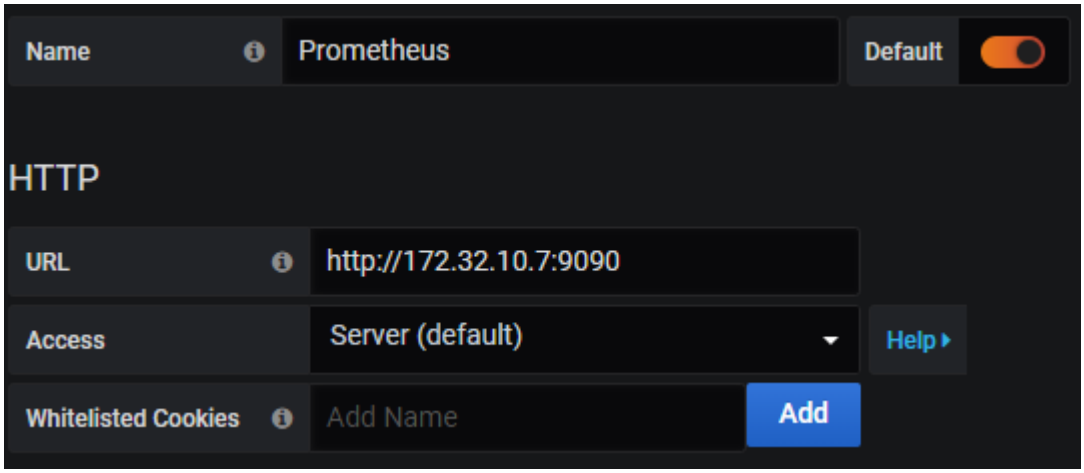
Zdroj: Vlastní zpracování

2.5.1. Nastavení zdroje dat

Vzhledem k faktu, že Grafana nedisponuje vlastním zdrojem dat, je potřeba zdroj dat nakonfigurovat. Tuto konfiguraci je možné provést pomocí tlačítka “Settings” a následně tlačítka “Data sources”. Dojde k přesměrování na stránku pro správu konfigurací. Pomocí tlačítka “Add data sources” lze poté přidat nový zdroj dat. Grafana poskytuje mimo Promethea mnoho dalších zdrojů, například:

- InfluxDB,
- Elasticsearch,
- MySQL,
- PostgreSQL,
- Azure monitor,
- Oracle.

Pro potřeby tohoto projektu je potřeba nakonfigurovat data pro Prometheus, obdobně jako je to předvedeno na následujícím obrázku (Obrázek 24):



Obrázek 24 – Grafana – Konfigurace zdroje

Zdroj: Vlastní zpracování

- URL – IP adresa a port, na kterém je vystavený nástroj Prometheus.
- Access – Definuje, jak budou zpracovávány požadavky na zdroj dat.
    - Server – Všechny požadavky budou předány z prohlížeče na backend Grafana server až poté bude předána zdroji dat.
    - Browser – Všechny požadavky budou předány přímo z prohlížeče ke zdroji dat.

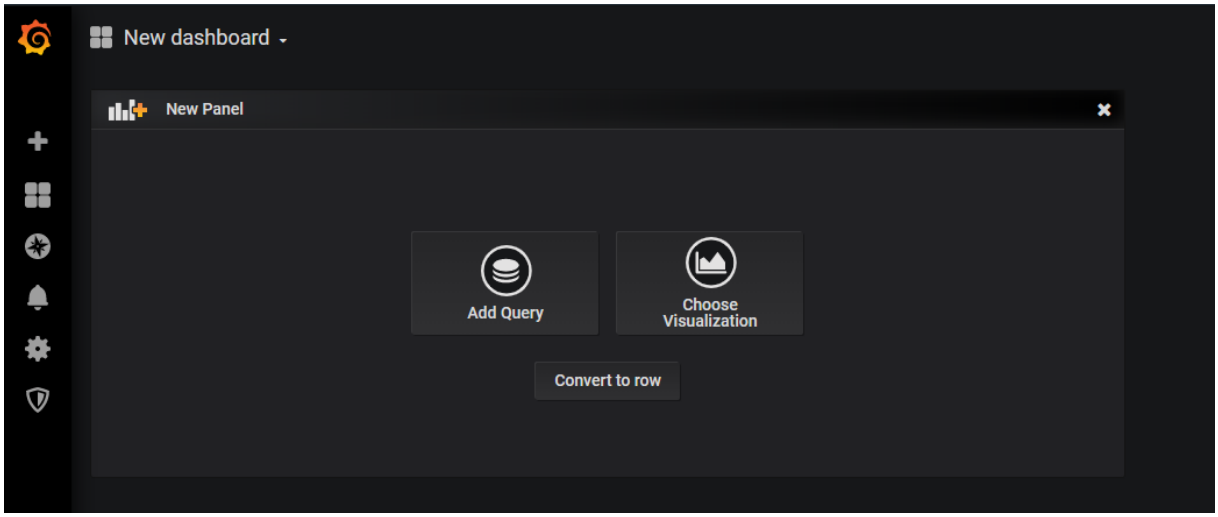
Při uložení konfigurace bude provedena validace, zda je zdroj dat dostupný a validní. Pokud vše proběhlo v pořádku, dojde k uložení. Po úspěšném provedení předchozích kroků je možné přejít k vytváření a návrhu vizualizací a upozorňovacích pravidel.

2.5.2. PromQL

Jazyk použitý pro vytváření dotazů do systému se nazývá PromQL. Pod tímto označením se skrývá celý název Prometheus Query Language. Tento dotazovací jazyk slouží pro vytváření jednoduchých, ale velmi výkonných dotazů pro vytváření vizualizací a upozorňovacích pravidel. Jednou z nevýhod tohoto programovacího jazyka je nepodobnost s ostatními dotazovacími jazyky obdobného typu. Seznamování s jazykem je ze začátku časově náročné a je potřeba vyzkoušet mnoho dotazů a případů. (*QUERYING PROMETHEUS*, 2020; *PromQL tutorial for beginners and humans*, 2019)

2.5.3. Vytváření dashboardu

Pro vytvoření nového dashboardu je potřeba najít a kliknout na ikonu se znaménkem plus. Po kliknutí se objeví nabídka create, kde je potřeba následně vybrat možnost "Dashboard". Po volbě bude provedeno přesměrování na stránku s možností výběru, jak je zobrazeno na následujícím obrázku (Obrázek 25).



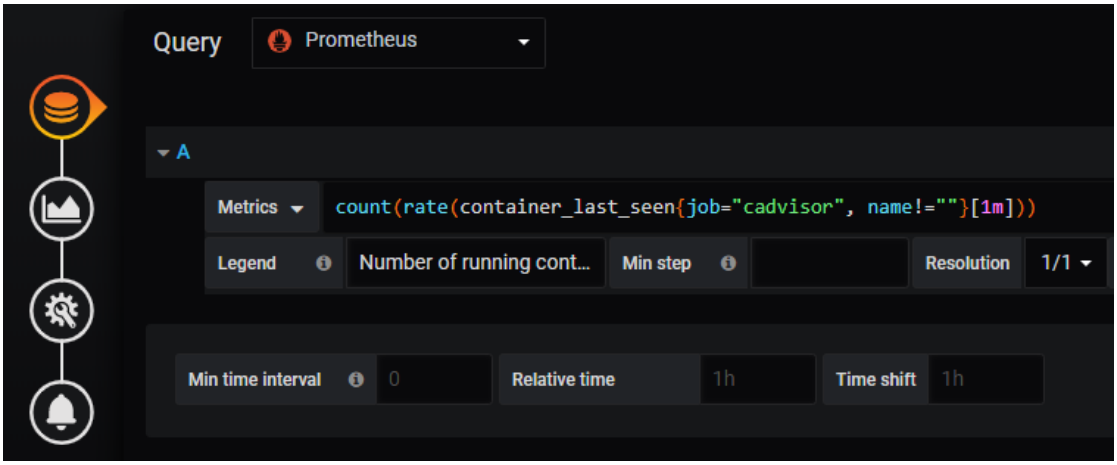
Obrázek 25 – Grafana – Rozcestník

Zdroj: Vlastní zpracování

Zobrazené možnosti vedou do stejného stavu, nicméně se liší postupem:

- Add Query – První z možností přesměruje požadavek přímo na výběr metriky. Jako typ vizualizace bude automaticky zvolen graf. V dalším kroku je možné tuto volbu změnit.
- Choose Visualization – Při zvolení této možnosti dojde k přesměrování na výběr dostupných vizualizací. Nicméně v následujícím kroku je potřeba vybrat metriku, která bude sledována.

Na obrázku (Obrázek 26) je uveden nakonfigurovaný dashboard, pro získání dat. Konfigurace dashboardu je rozdělena na 4 kroky, přičemž detailní popis kroku č. 2. bude vynechán. Tento krok není v tomto projektu z hlediska funkčnosti podstatný, neboť se zaměřuje především na grafickou stránku vizualizace.



Obrázek 26 – Grafana – Vytváření Dashboardu

Zdroj: Vlastní zpracování

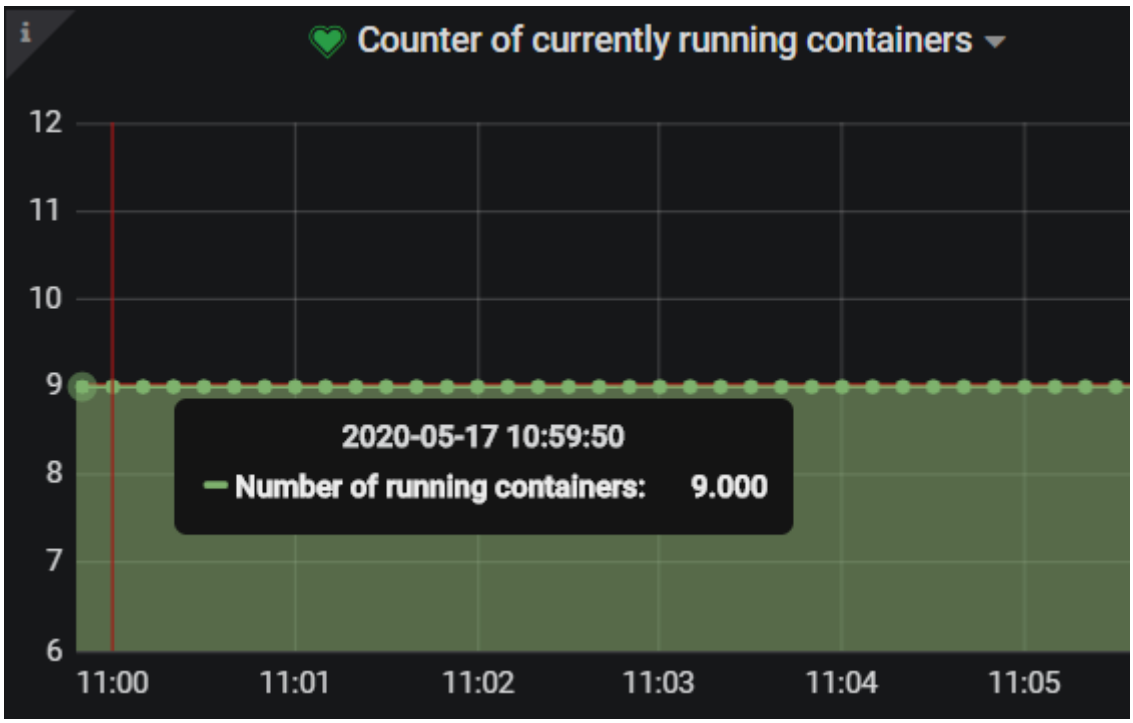
Konfigurace Dashboardu obsahuje:

- Query – Zdroj dat, ze kterých bude získávána metrika. Jedná se o ten zdroj, který byl zmíněn v kapitole 2.5.1.
- A – Identifikátor dotazu. Pokračuje dále podle abecedního pořadí.
- Metrics – count(rate(container_last_seen{job="cadvisor", name!=""}[1m]))
    - Dotaz do databáze pomocí jazyk PromQL.
        - Container_last_seen – Poslední dostupný časový otisk kontejneru. Obsahuje další rozšíření pro ještě jemnější dělení.
            - job=“cadvisor“ – Omezení názvem nástroje, který metriku získal.
            - name !=““ – Metrika musí obsahovat název kontejneru, nesmí být prázdná.
            - [1m] – Časový úsek, pro který jsou data vyhodnocována. Jedná se o časovou řadu. Lze si představit jako: Aktuální čas – 1 m.
        - count() – Získá počet výskytů daného pod dotazu.

- rate([1m]) – Výpočet průměrného nárůstu časové řady za daný časový interval.

- Legend – Specifikace hodnot zobrazených v grafu neboli legenda
- Min step – Jedná se o minimální rozmezí pro získání aktuální hodnoty. Výchozí nastavení je 5 s.

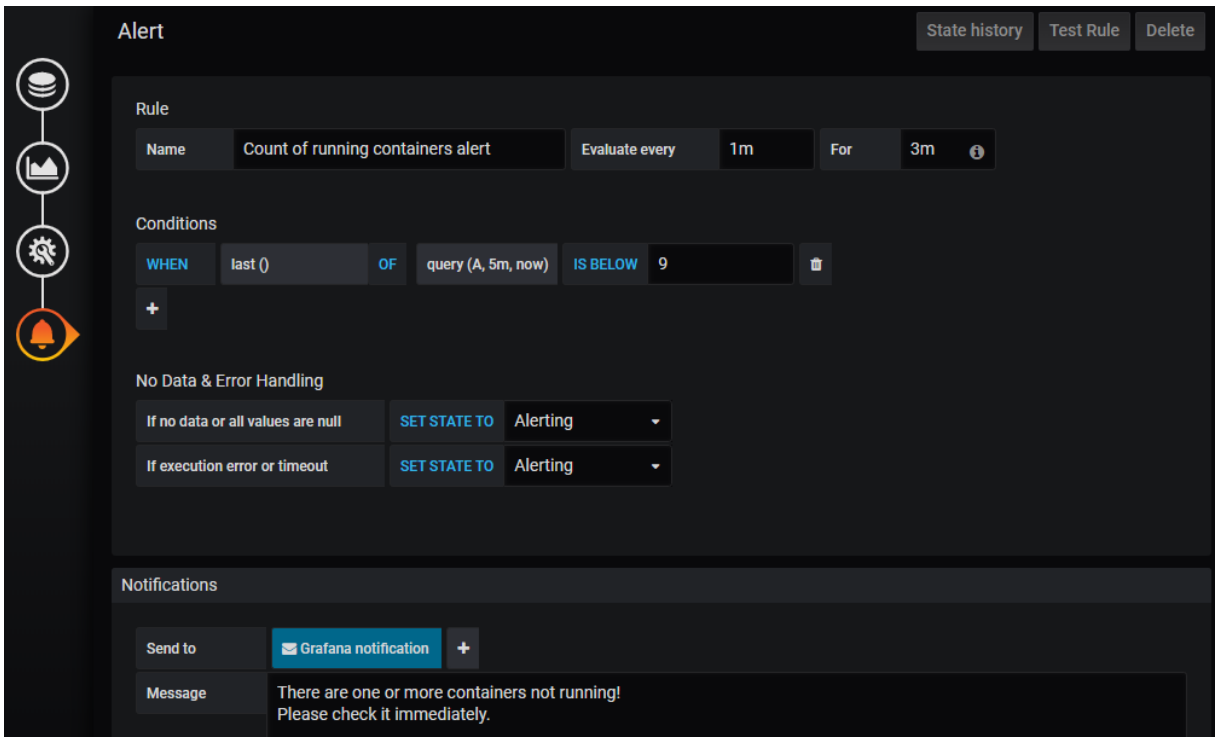
Ostatní, výše nezmíněné hodnoty, pouze upravují grafickou stránku vytvořeného grafu. Nejsou tedy z hlediska funkčnosti zmiňovány. Na následujícím obrázku (Obrázek 27) je zobrazen výsledek předchozí konfigurace.



Obrázek 27 – Grafana – Výsledný graf

Zdroj: Vlastní zpracování

Pozorný čtenář z předchozích kapitol pochopil, že nyní je monitorován aktuální počet běžících kontejnerů. V tomto projektu je počet všech běžících kontejnerů roven 9. Posledním, nicméně velmi důležitým krokem, je nastavení posílání notifikací při selhání.



Obrázek 28 – Grafana – Nastavení upozorňovacích pravidel

Zdroj: Vlastní zpracování

Na obrázku výše (Obrázek 28) je zobrazena základní konfigurace pro zasílání informativních notifikací při selhání jednoho nebo více kontejnerů. Jsou vyhodnocovány celkem 3 stavy (OK, PENDING, ALERTING). Stav OK napovídá o tom, že aktuální stav je v pořádku. Při stavu ALERTING dochází k odesílání upozorňovací notifikace. Stav PENDING je prodleva mezi stavy OK a ALERTING, při které může docházet k opětovnému vyhodnocování pravidel. Detailně rozebraná konfigurace je popsaná níže:

- Name – Název pravidla
    - Evaluate every – Při změně stavu z OK na stav PENDING (například v případě, že dojde ke snížení stavu běžících kontejnerů z 9 na 7), bude pravidlo vyhodnoceno ještě 3krát po dobu 3 minut, než dojde ke změně stavu na ALERTING a odeslání informativní notifikace. Tato vlastnost přidává prodlevu mezi stavy OK a ALERTING, po kterou zde figuruje pravděpodobnost, že dojde k nápravě. V rámci tohoto příkladu k opětovnému nastartování všech kontejnerů.
- Conditions – Definování podmínky, jejíž porušení provede změnu stavu.
    - WHEN {dostupné agregační funkce} OF {Identifikátor dotazu + úsek časové řady} {interval}

        - Dostupné agregační funkce – Grafana poskytuje základní agregační funkce pro vyhodnocování pravidel (avg, min, max, sum, count, median).
        - Identifikátor dotazu – Jedná se o identifikátor dotazu, který byl nakonfigurován v kapitolách zmíněných výše.
        - Úsek časové řady – Délka časového úseku, po který bude pravidlo vyhodnocováno.
        - Interval – Představuje podmínku, která určuje, v jakém intervalu se má hodnota nacházet (může být pod, nad, v intervalu nebo mimo interval).

- No Data & Error Handling
    - If no data or all values are null – Určuje, jaký bude stav, pokud nebudou data dostupná (OK, ALERTING nebo zapamatování posledního stavu).
    - If execution error or timeout – Nastavuje stav, který vznikne, jestliže při provádění dotazu nastane chyba nebo je překročen časový limit.
- Notifications
    - Send to – Určuje, jaký komunikační kanál bude využit pro zaslání notifikace.
    - Message – Zpráva, která bude přijata prostřednictvím nastaveného komunikačního kanálu.

2.5.4. Přidání komunikačního kanálu

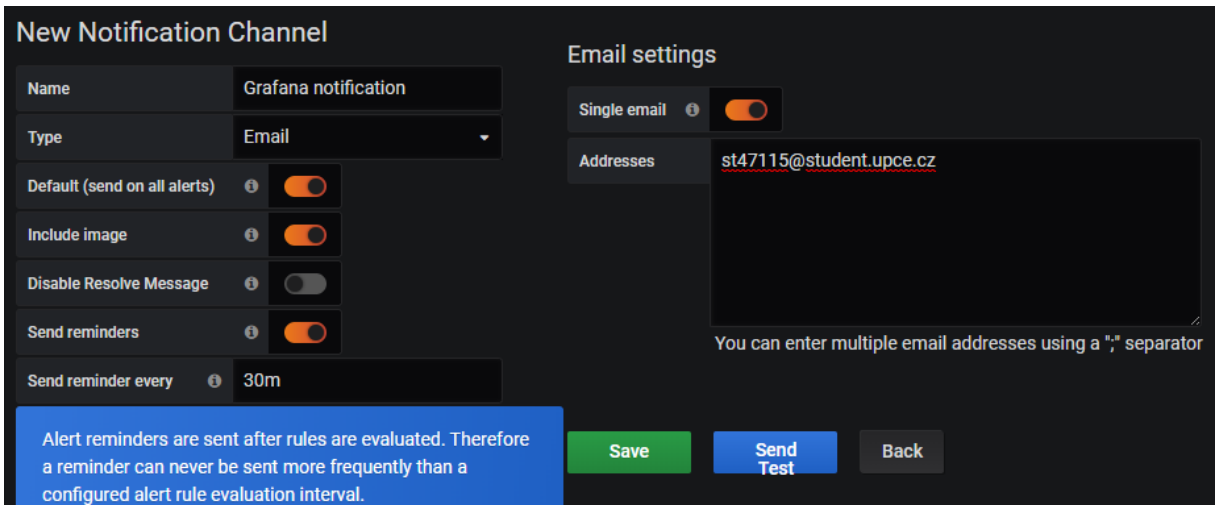
Nástroj Grafana podporuje několik druhů komunikačních kanálů pro zasílání informačních notifikací. Jmenovitě: Discord, E-mail, Slack, PagerDuty, Microsoft Teams. V tomto projektu bude využito zasílání notifikací pomocí e-mailu. V konfiguračním souboru grafana.ini je nezbytné vyplnit sekci SMTP. Na obrázku níže (Obrázek 29), je předvedena SMTP konfigurace. Po překonfigurováni je potřeba restartovat kontejner Grafana.

```
#################################### SMTP / Emailing ##########################
[smtp]
enabled = true
host = smtp.sendgrid.net:25
user = apikey
# If the password contains # or ; you have to wrap it with triple quotes. Ex """#password;"""
password = SG.b3B_9o
;cert_file =
;key_file =
skip_verify = true
from_address = st47115@student.upce.cz
from_name = Grafana monitoring
# EHLO identity in SMTP dialog (defaults to instance_name)
;ehlo_identity = dashboard.example.com
```

Obrázek 29 – Grafana – Nastavení SMTP Serveru

Zdroj: Vlastní zpracování

V dalším kroku je nezbytné nastavit nový komunikační kanál. Pro jeho vytvoření musíme vybrat z nabídky možnost s ikonou zvonku. Poté, co bude zobrazeno vyskakovací okno, zvolit volbu „Notification channels“ a následně zvolit možnost „New channel“. Pro lepší představu, je níže na obrázku (Obrázek 30) zobrazen již vytvořený notifikační kanál. Podrobněji bude rozebrán v komentářích pod obrázkem.



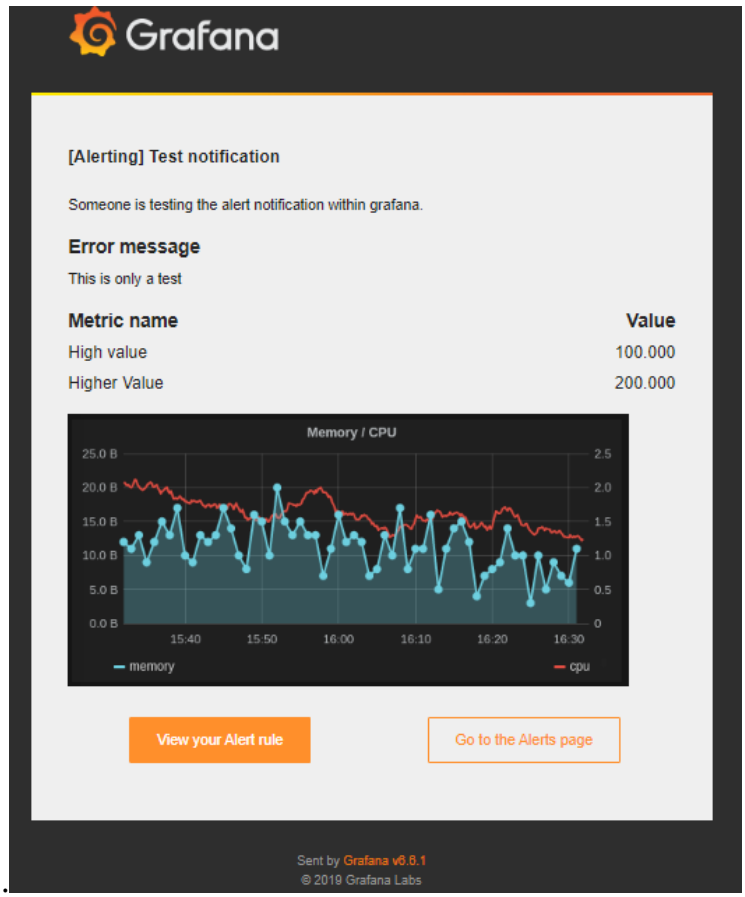
Obrázek 30 – Grafana – Přidání komunikačního kanálu

Zdroj: Vlastní zpracování

- Name – Jedná se o název notifikace.
- Type – Představuje druh informačního kanálu (E-mail, Discord, apod.).
- Default (send on all alerts)

- Include image – K odesílanému e-mailu bude připojen obrázek zobrazující aktuální stav.
- Disable Resolve Message – Poskytuje možnost vypnout notifikaci při změně stavu z ALERTING na OK.
- Send Reminders – Nastaví, zda budou zasílány připomínající notifikace.
    - Send reminders every – Definuje, jak často budou zasílány upomínající notifikace.
- Single email – Určuje, zda bude zaslán separátní e-mail pro všechny příjemce nebo bude odeslán hromadný mail.
- Addresses – Seznam e-mailových adres, pro které je notifikace určena. Musí být odděleny středníkem.

Po nakonfigurování je možné odeslat testovací notifikaci pomocí tlačítka „Send Test“. Pokud je všechno nakonfigurované v pořádku, na adresu, která byla nakonfigurována, by měla dorazit informativní notifikace, jak je předvedeno na obrázku (Obrázek 31). Data zobrazená v notifikaci jsou náhodně vygenerovaná a slouží pouze k ověření komunikace mezi nástrojem Grafana a e-mailovým klientem.



Obrázek 31 – Grafana – Přijmutí informativní notifikace

Zdroj: Vlastní zpracování

Toto byl poslední nezbytný konfigurační krok, v další kapitole bude provedeno ověření celé konfigurace a monitorovacích pravidel.

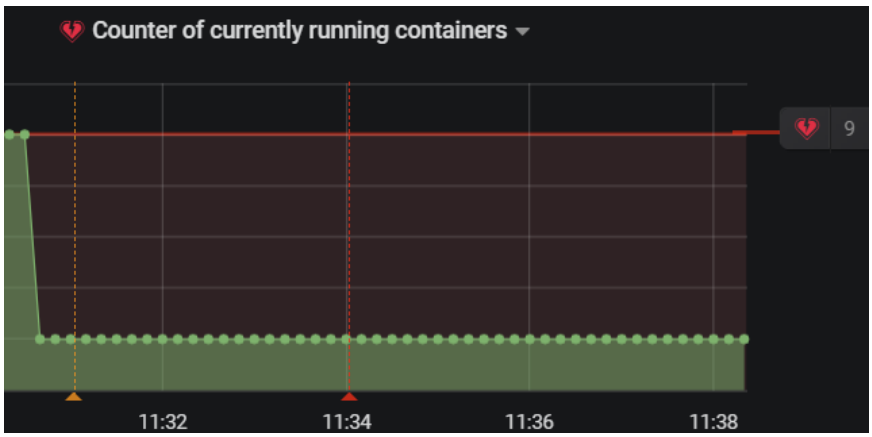
2.5.5. Ověření konfigurace a monitorovacích pravidel

Pro ověření správnosti a celistvosti konfigurace bude pro pravidlo, které bylo nastaveno v kapitole 2.5.3, ukončen jeden z běžících kontejnerů. Upozornění na vzniklou situaci bude zasláno na nakonfigurovaný e-mail.

Pro násilné ukončení kontejneru využijeme následující příkaz:

docker kill {identifikátor kontejneru}

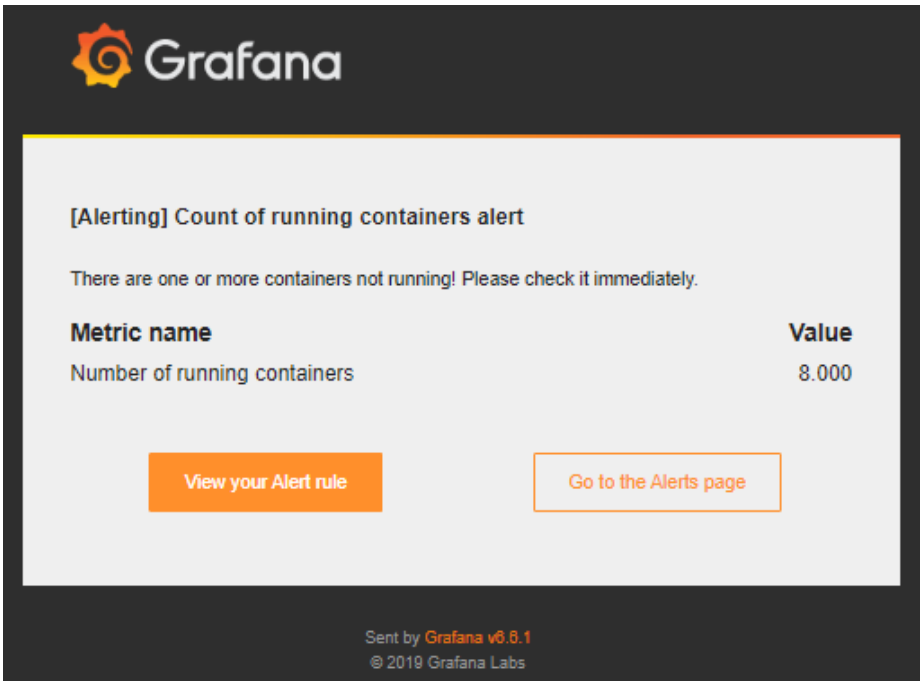
Pro lepší představu, je na dalším obrázku (Obrázek 32) předveden vývoj stavu při zjištění nefunkčnosti jednoho nebo více kontejnerů. Oranžová přerušovaná čára představuje změnu stavu z OK na PENDING. Červená přerušovaná čára značí přechod ze stavu PENDING na ALERTING.



Obrázek 32 – Grafana – Přechod mezi stavy

Zdroj: Vlastní zpracování

Následně po vyhodnocení pravidel dojde k odeslání notifikace na nastavený e-mail, jak je předvedeno na následujícím obrázku (Obrázek 33).



Obrázek 33 – Grafana – Upozorňovací notifikace

Zdroj: Vlastní zpracování

Celé prostředí je nyní nakonfigurované a může být jednoduše rozšířené o další monitorovací a upozorňovací pravidla, popřípadě další nástroje, které mohou sloužit pro získání a vystavování metriky. Vzhledem k tomu, že je možné všechny vytvořené dashboardy a pravidla vyexportovat, tak celé prostředí disponuje vysokou mírou přenositelnosti. Je tedy velmi jednoduché, rychlé a efektivní využít připravenou konfiguraci, například z testovacího prostředí, a nasadit ji na další nezávislé prostředí, například před produkční.

ZÁVĚR

Při poskytování spolehlivých IT služeb je nezbytnou součástí disponovat kvalitním monitorovacím systémem. Monitorovací systém zajistí nejen pravidelné sledování stavu prostředí a reporting v případě nalezeného problému, ale také má významný vliv na fungování provozovaného systému jako celku. Kvalitně navržený monitorovací systém je zodpovědný za správu technologií, které jsou v daném případě využívány. Před samotným výběrem monitorovacího systému je nutné si uvědomit, že zamýšlený projekt klade různá specifika a požadavky na dostupnost a výkon aplikací. Pro jednotlivé projekty se tedy hodí jiný typ monitorovacího systému. Hlavní doménou monitorovacího systému je detekování případných incidentů a v případě, že incident nastane, je rozpoznán velmi rychle. Tím se výrazně zvýší dostupnost sledovaného softwaru. Dále monitoring může velmi pozitivně ovlivnit vztahy se zákazníkem. Příznivější situace nastane, pokud poskytovatel služeb informuje zákazníka o nastalém problému s již navrhnutým řešením, než když zákazník informuje poskytovatele.

Již při vývoji aplikací a přípravě prostředí je nutné si uvědomit, že nikdo není neomylný. Ať už se jedná o programátora, databázového specialistu nebo administrátora. Problémy způsobené chybou v aplikaci, závadou na hardwarové infrastruktuře nebo nezamýšlenou chybou při servisních procedurách se vyskytují ve všech oblastech informačních technologií. Vzhledem k těmto skutečnostem je vhodné zahájit přípravu na monitoring již v průběhu vývoje softwaru. Je důležité již v této fázi uvažovat o různých možnostech, jak bude výsledný systém monitorován a popřípadě poskytnout definovaná rozhraní, přes která bude možné daný software monitorovat.

Cílem této práce bylo nalézt a nastavit vybraný dohledový systém pro sledování aplikací v podnikovém informačním systému. Sledovaná aplikace byla vystavěna a následně spuštěna jako kontejner pomocí virtualizační technologie Docker, obdobně jako všechny ostatní využívané nástroje. Pro sledování aplikace byl zvolen monitorovací nástroj Prometheus, který byl pro účel této práce nejvhodnější. Důvodem byl požadavek na monitorování aplikace spuštěné v kontejneru a rozšíření o dva exportéry. Monitoring hardwarových metrik nasazených kontejnerů zajišťuje exportér Cadvisor. Následně bylo nutné zprovoznit a nakonfigurovat nástroj Node exporter, který je zodpovědný za monitorování hardwarových metrik hostitelského zařízení, na kterém je technologie Docker spuštěna. Pro získání informací přímo ze spuštěné aplikace byly využity komponenty Spring Boot Actuator a Micrometer, které jsou součástí frameworku Spring Boot. Tyto komponenty poskytují předdefinované metriky a zároveň umožňují vytvářet vlastní metriky, které jsou exportovány ve formátu čitelném

pro Prometheus. Předposledním krokem bylo zprovoznění vizualizačního nástroje Grafana. Tento nástroj byl v této práci využit nejen pro vizualizaci, nýbrž i pro zasílání varovných notifikací. Po nakonfigurování všech potřebných komponent byl předveden scénář, který simuloval selhání jednoho z běžících kontejnerů a následné přijetí varovné notifikace, což vedlo k potvrzení funkčnosti navrhnutého systému.

Všechny využité komponenty jsou moderní technologie, které jsou aktuálně velmi populární a práce s nimi byla velice intuitivní. V praktické části práce jsou představeny jednotlivé kroky vedoucí k vytvoření rozšiřitelného monitorovacího prostředí. Navržené prostředí by bylo možné sestavit ještě více robustněji s využitím orchestrátoru kontejnerů, například nástrojem Kubernetes, se kterým je monitorovací systém Prometheus dobře provázán.

BIBLIOGRAFIE

ABOULLAITE, Mohammed, 2020. Spring boot metrics monitoring using TICK stack. *Aboullaite Med* [online]. Ghost [cit. 2020-04-23]. Dostupné z: https://aboullaite.me/spring-boot-metrics-monitoring-using-tick-stack/

Alertmanager, 2020. *GitHub* [online]. San Francisco: GitHub, Inc. [cit. 2020-04-19]. Dostupné z: https://github.com/prometheus/alertmanager

BERMAN, Daniel, 2020. Prometheus vs. Graphite: Which Should You Choose for Time Series or Monitoring?. *DZone* [online]. North Carolina, USA: AnswerHub [cit. 2020-04-19]. Dostupné z: https://dzone.com/articles/prometheus-vs-graphite-which-should-you-choose-for

BEZALEL, Amit, 2020. The docker age: Monitoring showdown Prom vs. TICK vs. Sensu. *Medium* [online]. USA: A Medium Corporation [cit. 2020-04-26]. Dostupné z: https://medium.com/@amit.bezalel/the-docker-age-monitoring-showdown-bda595b4b599

Cadvisor, 2020. *GitHub* [online]. San Francisco: GitHub, Inc. [cit. 2020-02-17]. Dostupné z: https://github.com/google/cadvisor

Collect Docker metrics with Prometheus, 2019. *Docker Documentation* [online]. San Francisco: Docker Inc. [cit. 2020-02-17]. Dostupné z: https://docs.docker.com/config/thirdparty/prometheus/

COMPARISON TO ALTERNATIVES, 2020. *Prometheus* [online]. San Francisco: Prometheus Authors [cit. 2020-03-01]. Dostupné z: https://prometheus.io/docs/introduction/comparison/

CONFIGURATION, © 2014-2020. *Prometheus* [online]. Prometheus Authors [cit. 2020-02-17]. Dostupné z: https://prometheus.io/docs/prometheus/latest/configuration/configuration/

DALLE VACCHE, Andrea, 2015. *Mastering Zabbix: Second Edition* [online]. 2 edition. Birmingham: Packt Publishing [cit. 2020-04-19]. ISBN 9781785289262. Dostupné z: https://subscription.packtpub.com/book/networking_and_servers/9781785289262

Distributed Monitoring, © 2001-2020. *Zabbix* [online]. Brooklyn, USA: Zabbix LLC. [cit. 2020-04-19]. Dostupné z: https://www.zabbix.com/distributed_monitoring

Docker overview, 2019. *Docker Documentation* [online]. San Francisco: Docker Inc. [cit. 2020-04-19]. Dostupné z: https://docs.docker.com/get-started/overview/

Dockerfile reference, 2019. *Docker Documentation* [online]. San Francisco: Docker Inc. [cit. 2020-01-29]. Dostupné z: https://docs.docker.com/engine/reference/builder/

ELK Stack Tutorial: Learn Elasticsearch, Logstash, and Kibana, 2020. *Guru99* [online]. Ahemdabad, Gujrat: Guru99 [cit. 2020-04-26]. Dostupné z: https://www.guru99.com/elk-stack-tutorial.html#5

ELLINGWOOD, Justin, 2020. An Introduction to Metrics, Monitoring, and Alerting. *DigitalOcean* [online]. New York, USA: DigitalOcean, LLC. [cit. 2020-04-19]. Dostupné z: https://www.digitalocean.com/community/tutorials/an-introduction-to-metrics-monitoring-and-alerting

EXPORTERS AND INTEGRATIONS, 2020. *Prometheus* [online]. San Francisco: Prometheus Authors [cit. 2020-04-19]. Dostupné z: https://prometheus.io/docs/instrumenting/exporters/

Forwarding Data, 2020. *Splunk>docs* [online]. San Francisco: Splunk Inc. [cit. 2020-04-26]. Dostupné z: https://docs.splunk.com/Documentation/Splunk/8.0.3/Forwarding/Typesofforwarders

Grafana, 2020. *Grafana Labs* [online]. Grafana Labs [cit. 2020-03-02]. Dostupné z: https://grafana.com/grafana/

Chronograf, 2020. *InfluxData* [online]. San Francisco: InfluxData Inc. [cit. 2020-04-23]. Dostupné z: https://www.influxdata.com/time-series-platform/chronograf/

Implementing Custom Service Discovery, 2020. *Prometheus* [online]. San Francisco: Prometheus Authors [cit. 2020-04-19]. Dostupné z: https://prometheus.io/blog/2018/07/05/implementing-custom-sd/

InfluxDB 1.8 documentation, 2020. *InfluxData* [online]. San Francisco: InfluxData Inc. [cit. 2020-04-23]. Dostupné z: https://docs.influxdata.com/influxdb/v1.8/

InfluxDB Enterprise, 2020. *InfluxData* [online]. San Francisco: InfluxData Inc. [cit. 2020-04-23]. Dostupné z: https://www.influxdata.com/products/influxdb-enterprise/

Integrations, 2020. *InfluxData* [online]. San Francisco: InfluxData Inc. [cit. 2020-04-23]. Dostupné z: https://www.influxdata.com/products/integrations/

JAISWAL, Sharad, 2020. What is ElasticSearch? Pros, Cons and Features List. *Courseya* [online]. Courseya.com [cit. 2020-04-26]. Dostupné z: https://www.courseya.com/blog/what-is-elasticsearch-pros-cons-and-features-list/

Kapacitor, 2020. *InfluxData* [online]. San Francisco: InfluxData Inc. [cit. 2020-04-23]. Dostupné z: https://www.influxdata.com/time-series-platform/kapacitor/https://www.influxdata.com/time-series-platform/kapacitor/

KOVACS, Kristof, © 2001-2020. Zabbix vs Nagios comparison. *Kristof Kovacs: Software architect, consultant* [online]. Budapest, Hungary: Kristof Kovacs [cit. 2020-04-19]. Dostupné z: https://kkovacs.eu/zabbix-vs-nagios

LOGSTASH: Centralize, transform & stash your data, 2020. *Elastic* [online]. California, USA: Elasticsearch B.V. [cit. 2020-04-26]. Dostupné z: https://www.elastic.co/logstash

Micrometer Prometheus, 2017. *MICROMETER: application monitoring* [online]. San Francisco: Pivotal Software, Inc. [cit. 2020-03-01]. Dostupné z: https://micrometer.io/docs/registry/prometheus

Monitoring služeb a mikroslužeb psaných v Go nástrojem Prometheus, 2020. *ROOT.CZ* [online]. Praha: Internet Info, s. r. o. [cit. 2020-03-01]. Dostupné z: https://www.root.cz/clanky/monitoring-sluzeb-a-mikrosluzeb-psanych-v-go-nastrojem-prometheus/#k06

Nagios Vs. Zabbix Vs. PRTG Vs. Spiceworks Vs. Solarwinds Network Performance Monitor, 2020. *IT Central Station* [online]. New York, USA: IT Central Station [cit. 2020-04-19]. Dostupné z: https://www.itcentralstation.com/product_reviews/zabbix-review-32935-by-it_user174738

Node exporter, 2020. *GitHub* [online]. San Francisco: GitHub, Inc. [cit. 2020-02-17]. Dostupné z: https://github.com/prometheus/node_exporter

Notifications & Automatic actions, © 2001-2020. *Zabbix* [online]. Brooklyn, USA: Zabbix LLC. [cit. 2020-04-19]. Dostupné z: https://www.zabbix.com/notification

Orientation and setup, 2019. *Docker Documentation* [online]. San Francisco: Docker Inc. [cit. 2020-04-19]. Dostupné z: https://docs.docker.com/get-started/

Overview of Docker Compose, 2019. *Docker Documentation* [online]. San Francisco: Docker Inc. [cit. 2020-01-30]. Dostupné z: https://docs.docker.com/compose/

Overview, © 2011-2017. *Graphite* [online]. The Graphite Project Revision [cit. 2020-04-19]. Dostupné z: https://graphiteapp.org

Overview, 2020. *Prometheus* [online]. San Francisco: Prometheus Authors [cit. 2020-04-19]. Dostupné z: https://prometheus.io/docs/introduction/overview/

Prometheus Pushgateway, 2020. *GitHub* [online]. San Francisco: GitHub, Inc. [cit. 2020-04-19]. Dostupné z: https://github.com/prometheus/pushgateway

Prometheus, 2020. *GitHub* [online]. San Francisco: GitHub, Inc. [cit. 2020-04-19]. Dostupné z: https://github.com/prometheus/prometheus

PromQL tutorial for beginners and humans, 2019. *Medium* [online]. USA: A Medium Corporation [cit. 2020-03-01]. Dostupné z: https://medium.com/@valyala/promql-tutorial-for-beginners-9ab455142085

QUERY EXAMPLES, 2020. *Prometheus* [online]. San Francisco: Prometheus Authors [cit. 2020-03-01]. Dostupné z: https://prometheus.io/docs/prometheus/latest/querying/examples/

QUERYING PROMETHEUS, 2020. *Prometheus* [online]. San Francisco: Prometheus Authors [cit. 2020-03-01]. Dostupné z: https://prometheus.io/docs/prometheus/latest/querying/basics/

Quick Guide to Micrometer, 2020. *Baeldung* [online]. București, Rumunsko: Baeldung SRL. [cit. 2020-03-01]. Dostupné z: https://www.baeldung.com/micrometer

SECELEANU, Cosmin, 2020. Collect and analyze Docker logs using Filebeat and Elastic Stack(ELK). *Medium* [online]. USA: A Medium Corporation [cit. 2020-04-26]. Dostupné z: https://medium.com/@sece.cosmin/docker-logs-with-elastic-stack-elk-filebeat-50e2b20a27c6

Spring Boot Actuator: Production-ready Features, 2020. *Spring* [online]. San Francisco: Pivotal Software, Inc. [cit. 2020-03-01]. Dostupné z: https://docs.spring.io/spring-boot/docs/current/reference/html/production-ready-features.html#production-ready-metrics

Spring Boot Actuator, 2020. *Spring* [online]. San Francisco: Pivotal Software, Inc. [cit. 2020-03-01]. Dostupné z: https://www.baeldung.com/spring-boot-actuators

Telegraf, 2020. *InfluxData* [online]. San Francisco: InfluxData Inc. [cit. 2020-04-23]. Dostupné z: https://www.influxdata.com/time-series-platform/telegraf/

The Carbon Daemons, © 2011-2017. *Graphite* [online]. The Graphite Project Revision [cit. 2020-04-19]. Dostupné z: https://graphite.readthedocs.io/en/latest/carbon-daemons.html#carbon-aggregator-py

The Whisper Database, © 2011-2017. *Graphite* [online]. The Graphite Project Revision [cit. 2020-04-19]. Dostupné z: https://graphite.readthedocs.io/en/latest/whisper.html

VARDHAN, 2020. Splunk Architecture: Tutorial On Forwarder, Indexer And Search Head. *Edureka!* [online]. Porto Alegre, Brasil: Brain4ce Education Solutions Pvt. [cit. 2020-04-26]. Dostupné z: https://www.edureka.co/blog/splunk-architecture/

Webapp Database Setup, © 2011-2017. *Graphite* [online]. The Graphite Project Revision [cit. 2020-04-19]. Dostupné z: https://graphite.readthedocs.io/en/latest/config-database-setup.html

What are Elasticsearch Beats?, 2020. *ObjectRocket* [online]. Texas, USA: ObjectRocket [cit. 2020-04-26]. Dostupné z: https://www.objectrocket.com/resource/what-are-elasticsearch-beats/

What is Splunk? Beginners Tutorial, 2020. *Guru99* [online]. Ahemdabad, Gujrat: Guru99 [cit. 2020-04-26]. Dostupné z: https://www.guru99.com/splunk-tutorial.html#5

What is Splunk?, © 2011-2020. *IntelliPaat* [online]. Bangalore, Karnataka: intellipaat.com [cit. 2020-04-26]. Dostupné z: https://intellipaat.com/blog/what-is-splunk/

What is Splunk?, 2020. *Stan Zhou's Hexo Technical Blog* [online]. Stan Zhou [cit. 2020-04-26]. Dostupné z: https://cwzhou.win/2019/07/15/splunk/

WHY YOU NEED A MONITORING SYSTEM: PROVIDE THE BEST SERVICE FOR YOUR CLIENTS, 2020. *PANDORA FMS: Monitoring blog* [online]. Madrid: PANDORA FMS [cit. 2020-04-19]. Dostupné z: https://pandorafms.com/blog/why-you-need-a-monitoring-system/

Zabbix Documentation 4.4, © 2001-2020. *Zabbix* [online]. Brooklyn, USA: Zabbix LLC. [cit. 2020-04-19]. Dostupné z: https://www.zabbix.com/documentation/current/manual

PŘÍLOHY

PŘÍLOHA A – Seznam použitého softwarového a hardwarového vybavení

Hardwarové vybavení:

- Dell Inspiron 7566

Softwarové vybavení:

- Linuxová distribuce CentOS verze 7.6.1810 (Core)
- Docker verze 1.13.1
- Prometheus verze 2.15
- Node Exporter 0.18.1
- Cadvisor 0.32.0
- Grafana 6.6.1