TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Využití softwarově definovaného úložiště pro nasazení technologií pro převod řeči na text

Bakalářská práce

Studijní program: B0613A140005 Informační technologie
Studijní obor: Inteligentní systémy

Autor práce: **Martin Motejlek**
Vedoucí práce: Ing. Ondřej Smola
Ústav informačních technologií a elektroniky

Liberec 2022

TECHNICKÁ UNIVERZITA V LIBERCI
Fakulta mechatroniky, informatiky
a mezioborových studií

Zadání bakalářské práce

Využití softwarově definovaného úložiště pro nasazení technologií pro převod řeči na text

Jméno a příjmení: **Martin Motejlek**
Osobní číslo: M19000028
Studijní program: B0613A140005 Informační technologie
Specializace: Inteligentní systémy
Zadávající katedra: Ústav informačních technologií a elektroniky
Akademický rok: **2021/2022**

Zásady pro vypracování:

1. Seznamte se s technologiemi softwarově definovaného úložiště na platformě Kubernetes.
2. S pomocí vybrané technologie implementujte řešení poskytující škálovatelnou a vysoce dostupnou vrstvu pro distribuci dat, které jsou využívána systémem pro rozpoznávaní řeči vyvíjeném na TUL.
3. Implementaci poté nainstalujte a otestujte na interním privátním clusteru.

Rozsah grafických prací:	dle potřeby dokumentace
Rozsah pracovní zprávy:	30-40
Forma zpracování práce:	tištěná/elektronická
Jazyk práce:	Čeština

Seznam odborné literatury:

[1] Lukša, Marko. *Kubernetes in Action*. Manning, 2017. ISBN 9781617293726.

[2] FISK, Nick. *Mastering Ceph: Infrastructure storage solutions with the latest Ceph release*. 2nd ed. Packt Publishing, 2019. ISBN 1789610702.

Vedoucí práce:	Ing. Ondřej Smola Ústav informačních technologií a elektroniky
Datum zadání práce:	12. října 2021
Předpokládaný termín odevzdání:	22. května 2023

L.S.

prof. Ing. Zdeněk Plíva, Ph.D.
děkan

prof. Ing. Ondřej Novák, CSc.
vedoucí ústavu

V Liberci dne 19. října 2021

Prohlášení

Prohlašuji, že svou bakalářskou práci jsem vypracoval samostatně jako původní dílo s použitím uvedené literatury a na základě konzultací s vedoucím mé bakalářské práce a konzultantem.

Jsem si vědom toho, že na mou bakalářskou práci se plně vztahuje zákon č. 121/2000 Sb., o právu autorském, zejména § 60 – školní dílo.

Beru na vědomí, že Technická univerzita v Liberci nezasahuje do mých autorských práv užitím mé bakalářské práce pro vnitřní potřebu Technické univerzity v Liberci.

Užiji-li bakalářskou práci nebo poskytnu-li licenci k jejímu využití, jsem si vědom povinnosti informovat o této skutečnosti Technickou univerzitu v Liberci; v tomto případě má Technická univerzita v Liberci právo ode mne požadovat úhradu nákladů, které vynaložila na vytvoření díla, až do jejich skutečné výše.

Současně čestně prohlašuji, že text elektronické podoby práce vložený do IS/STAG se shoduje s textem tištěné podoby práce.

Beru na vědomí, že má bakalářská práce bude zveřejněna Technickou univerzitou v Liberci 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.

Jsem si vědom následků, které podle zákona o vysokých školách mohou vyplývat z porušení tohoto prohlášení.

Martin Motejlek

Poděkování

Rád bych poděkoval vedoucímu Ing. Ondřeji Smolovi za odborné vedení práce, poskytnuté prostředky, připomínky a rady. Dále děkuji rodičům za podporu a pomoc při korektuře práce.

Využití softwarově definovaného úložiště pro nasazení technologií pro převod řeči na text

Abstrakt

Tato bakalářská práce se zabývá využitím spolehlivého distribuovaného úložiště Ceph v rámci platformy pro přepis řeči na text vyvíjené na Fakultě mechatroniky, informatiky a mezioborových studií TUL.

V rámci práce je popsáno úložiště Ceph a vybrané komponenty, které s ním v systému souvisí: zejména systém Kubernetes pro automatizaci správy distribuovaného systému a nástroje Prometheus a Grafana Loki, které sbírají diagnostická data a Ceph využívají k jejich ukládání.

Dále byla v rámci práce vyvinuta aplikace, která ze systému stáhne diagnostická data o průběhu zadaného uživatelského požadavku na přepis a vytvoří z nich výstup vhodný pro analýzu člověkem. Byla navržena vhodná interpretace dostupných údajů o požadavku za účelem stanovení stahovaného časového období. Pro komunikaci s rozhraními nástrojů Prometheus a Loki byly implementovány vlastní klientské knihovny. Dále byly navrženy dotazy pro získání požadovaných dat ve vhodně podobě. Pomocí dávkového dotazování bylo vyřešeno stažení neomezeného množství logu. Výsledná aplikace je ovládána z rozhraní v příkazové řádce a její výstup má formu archivu, který obsahuje mimo jiné HTML dokument s interaktivními grafy metrik. Běh aplikace byl formou smoke testu úspěšně ověřen vůči nasazení systému na testovacím clusteru.

Na závěr práce byla nasazena privátní instance systému pomocí připraveného, ale ne zcela odladěného automatizovaného řešení využívajícího nástroj Ansible. V této části je popsáno řešení vybraných problémů, které při jeho použití nastaly.

Klíčová slova: cloud, Kubernetes, distribuované úložiště Ceph, dotazování Prometheus, dotazování Grafana Loki

Use of software-defined storage for the deployment of speech-to-text technologies

Abstract

This bachelor's thesis deals with the use of a reliable distributed storage called Ceph as part of the speech-to-text platform developed at the Faculty of Mechatronics, Informatics and Interdisciplinary Studies, TUL.

The text describes the Ceph storage and selected related components: Kubernetes for the automation of distributed system administration, and Prometheus and Grafana Loki, which collect diagnostic data and save it into Ceph.

As part of the thesis, an application was developed which downloads data describing the course of a user speech-to-text request and outputs it in a form appropriate for analysis by a human. An appropriate interpretation of available information about the request was designed in order to establish the time interval for download. In order to communicate with the interfaces of Prometheus and Loki, custom client libraries were implemented. Queries were designed in order to obtain required data in an appropriate form. The problem of acquiring an unlimited amount of log was solved by batched querying. The resulting application has a command line interface and outputs data in the form of an archive, which contains an HTML document with interactive graphs of metrics, among other things. The application was successfully tested using a smoke test against an instance of the system on a testing cluster.

Finally, a private instance of the system was deployed using an automated but not fully debugged solution that used Ansible. This part of the thesis describes selected problems which arose during the deployment and their solutions.

Keywords: cloud, Kubernetes, distributed storage Ceph, querying Prometheus, querying Grafana Loki

Obsah

Seznam obrázků

Seznam zkratek

CephFS Ceph File System

CNCF Cloud Native Computing Foundation

MDS Ceph Metadata Server

MGR Ceph Manager

MON Ceph Monitor

OSD Ceph Object Storage Daemon

TUL Technická univerzita v Liberci

Úvod

Kontextem této bakalářské práce je cloudový systém vyvíjený na Fakultě mechatroniky, informatiky a mezioborových studií TUL, který poskytuje službu přepisu řeči na text (tento systém je dále označován „systém pro přepis“). Principem této služby je, že jí uživatel vysílá (stream) zvukovou nahrávku mluveného projevu a služba vysílá zpět její textový přepis. Vypracování této práce probíhalo v rámci aktivně vyvíjené nové verze.

Typickým požadavkem na cloudovou službu je tzv. vysoká dostupnost, tedy že systém nepodléhá výpadkům a přetížením. Pro splnění této vlastnosti mívají cloudové služby charakter distribuovaného systému sestávajícího se z tzv. mikroslužeb – aplikací plnících dílčí funkce systému. Instance mikroslužeb jsou nasazeny na skupině strojů, tzv. uzlů, nazývané cluster. V rámci ní jsou uzly propojeny sítí, přes kterou spolu instance mikroslužeb komunikují. Pro úplnost je třeba dodat, že slovem „cluster“ je rovněž často označována skupina instancí aplikací tvořících určitý systém, jež jsou v tomto kontextu označovány „uzly“ – tato označení jsou používána například u systému Ceph popisovaného v tomto textu.

Architektura mikroslužeb umožňuje docílit vysoké dostupnosti např. díky podpoře horizontálního škálování – efektivního zvyšování výkonu nebo propustnosti systému prostým přidáváním strojů a replikací komponent – a možnosti rozmístit repliky komponent a dat na více uzlů clusteru, čímž se zvyšuje odolnost proti výpadkům. Nasazené aplikace ale musí být naprogramovány tak, aby byly schopné těchto možností využít. Zároveň musí uspokojivě odolávat problémům distribuovaných systémů spojených s nespolehlivostí síťového spojení a samotných uzlů.

Tématem této bakalářské práce je využití softwarově definovaného úložiště v systému pro přepis. Pro spolehlivé ukládání trvalých dat v rámci distribuovaného systému je potřeba speciálně navržené úložiště; v systému pro přepis zastává tuto roli řešení s názvem Ceph. Toto úložiště je stejně jako ostatní komponenty systému pro přepis nasazeno na platformě Kubernetes, která automatizuje nasazování aplikací do clusteru a řešení výpadků. Konkrétním využitím úložiště, kterým se tato práce zabývá, je ukládání diagnostických dat, v tomto případě metrik a logu, které jsou z celého clusteru průběžně sbírány za účelem odhalování a diagnostiky problémů.

Cíle práce lze rozdělit do tří bodů:

1. Seznámit se se softwarově definovaným úložištěm Ceph a jeho využitím v systému pro přepis.

2. Vyvinout aplikaci, která umožní ze systému stahovat uložená diagnostická data o průběhu vykonávání požadavků uživatelů.

3. Zprovoznit privátní nasazení systému a otestovat oproti němu běh vyvinuté aplikace.

Pro vypracování práce bylo k dispozici prostředí testovacího clusteru umístěného na TUL, na kterém byl systém pro přepis nasazen. Toto prostředí bylo sdíleno s dalšími vývojáři. Pro privátní nasazení byly poskytnuty potřebné hardwarové prostředky.

První část práce popisuje systém pro přepis, jeho komponenty a použité technologie včetně úložiště Ceph a způsob, jakým je systém pro přepis nasazován. Vzhledem k rozsahu systému pro přepis a používaných technologií bylo nutné jejich popis omezit na aspekty, které se týkají cílů práce. Druhá část se věnuje vývoji aplikace pro stahování diagnostických dat ukládaných systémem. V třetí části je popsán průběh nasazení systému pro přepis za účelem otestování vyvinuté aplikace a jeho výsledky.

Pro popis systému pro přepis byly využity informace z konzultací s vedoucím bakalářské práce, který je jeho vývojářem, kódu jádra systému (viz 1.1), kódu řešení pro automatizaci nasazení systému (viz 3.2.1) a webových rozhraní komponent z nasazené instance systému na testovacím clusteru. V průběhu vypracovávání bakalářské práce byl systém pro přepis aktivně vyvíjen a měnil se včetně svého rozhraní, čemuž se musela práce přizpůsobovat. Popis systému v textu práce reflektuje stav v březnu/dubnu 2022.

V tabulce 1 jsou uvedeny verze nejpodstatnějších technologií třetích stran, ke kterým byla práce vypracována. Všechny uvedené technologie jsou v průběhu textu představeny.

Tabulka 1: Verze nejpodstatnějších technologií třetích stran popisovaných v textu práce

Technologie	**Verze**
Ansible	5
Ceph	pacific
Go	1.17
Grafana	8.3
Kubernetes	1.23
Loki	2.4
MAAS	3.1
Prometheus	2.33
Rook	1.8

1 Systém pro přepis řeči na text

V první části práce je podrobněji popsán systém pro přepis, jenž byl představen v úvodu bakalářské práce. Vzhledem k rozsahu tohoto systému a používaných technologií je jejich popis omezen na aspekty související s cíli práce.

Nejdříve je představena architektura systému pro přepis a podrobněji vysvětleny jeho vybrané součásti. Následně je přiblíženo, jakým způsobem je systém pro přepis nasazován. Na závěr je krátce zmíněn programovací jazyk Go, který je při vývoji systému pro přepis používán.

1.1 Přehled vybraných komponent systému pro přepis

Systém pro přepis se skládá z komponent vyvíjených třetími stranami a z komponent, které jsou vyvíjeny jeho autory přímo pro potřeby systému.

Základ systému pro přepis tvoří systém **Kubernetes** (viz 1.2), který automatizuje nasazení a správu kontejnerizovaných aplikací v clusteru. V rámci něj jsou nasazeny ostatní komponenty.

Součástí systému pro přepis je spolehlivé distribuované úložiště **Ceph** (viz 1.3), které v clusteru zastává role sdíleného souborového systému a objektového úložiště s rozhraním S3. Mimo jiné jsou do něj ukládána diagnostická data, která jsou ze systému průběžně sbírána. Do Kubernetes je Ceph nasazen pomocí nástroje **Rook**.

K získávání diagnostických dat ze systému pro přepis slouží nástroje **Prometheus** (viz 1.4), který sbírá a ukládá metriky, a **Grafana Loki** (také označován pouze „**Loki**“) (viz 1.5), který sbírá a ukládá logy. Pro interaktivní vizualizaci sesbíraných diagnostických dat je v systému nasazena **Grafana**[1] (viz 1.6).

Samotný přepis řeči na text a obsluhu uživatelských požadavků obstarává komponenta, která je v rámci tohoto textu nazývána „**jádro systému pro přepis**“ a jíž je věnována kapitola 1.7.

V systému pro přepis může být volitelně nasazen open-source registr **Harbor**, jehož vývoj je zaštiťován Cloud Native Computing Foundation (CNCF)[2] (*Harbor*, ©2022) a který v rámci systému pro přepis slouží jako repozitář obrazů komponent systému a tzv. „**služeb pro přepis**“, což jsou programy využívané systémem pro přepis k realizaci přepisu řeči na text.

[1] „Grafana“ a „Grafana Loki“ (také jen „Loki“) jsou dva různé nástroje.

[2] Cloud Native Computing Foundation (CNCF) je nadace zaštiťující řadu open-source projektů využívaných v cloudových systémech (*Cloud Native Computing Foundation*, ©2022).

1.2 Kubernetes: orchestrace kontejnerizovaných aplikací

Kubernetes je open-source systém vyvíjený pod záštitou CNCF sloužící k automatizaci nasazení a správy kontejnerizovaných aplikací v clusteru (*Kubernetes Documentation*, ©2022, Home). Základním principem jeho fungování je, že je definován požadovaný stav clusteru, ke kterému se Kubernetes snaží skutečný stav clusteru v průběhu času přibližovat. Kubernetes tak automaticky reaguje např. na výpadky uzlů, pády aplikací nebo změny v požadovaném stavu.

Jedním ze základních nástrojů pro práci s Kubernetes je program kubectl, který umožňuje z rozhraní příkazové řádky ovládat Kubernetes cluster a vypisovat jeho vlastnosti (*Kubernetes Documentation*, ©2022, Command line tool (kubectl)).

1.2.1 Základní koncepty Kubernetes

Jednu instanci aplikace nasazené v Kubernetes reprezentuje tzv. pod. Dokumentací je charakterizován jako nejmenší jednotka, kterou lze v Kubernetes nasadit a spravovat. Je tvořen jedním nebo více kontejnery. Pody jsou systémem Kubernetes automaticky umisťovány na uzly clusteru (node), viz obrázek 1.1. Každému podu je v rámci privátní sítě Kubernetes clusteru přidělena IP adresa. (*Kubernetes Documentation*, ©2022, Pods)

Různé aspekty Kubernetes clusteru jsou s různou mírou abstrakce popisovány pomocí tzv. objektů (objects). Typy objektů (kind) jsou například:

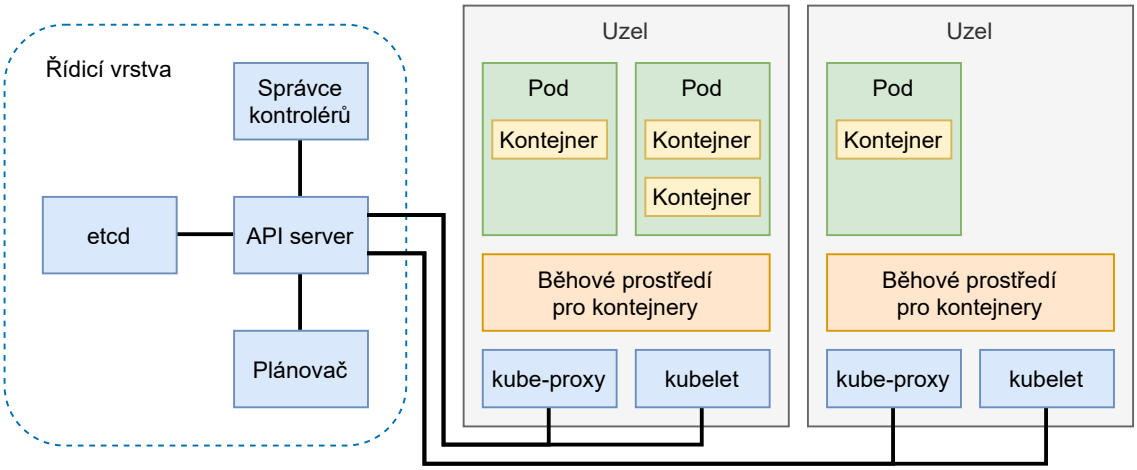
- pody;
- uzly clusteru;
- nasazení (Deployment), pomocí kterých uživatel definuje nasazení aplikace, viz kapitola 1.2.3;
- úložiště a jiné zdroje přiřazené podům, viz kapitola 1.2.3.

Každý objekt má uživatelem definovaný název. Většina objektů obsahuje specifikaci svého požadovaného a aktuálního stavu. Objekty lze označit štítky (labels), které při konfiguraci slouží pro výběr skupin objektů s určitou vlastností. Objekty lze také rozdělit do jmenných prostorů (namespaces), např. podle toho, se kterou komponentou systému souvisí. Názvy objektů jsou unikátní pouze v rámci jmenného prostoru. (*Kubernetes Documentation*, ©2022, Understanding Kubernetes Objects, Pods, Nodes, Labels and Selectors, Namespaces)

Řízení Kubernetes je, dle dokumentace, založeno na principu řídicích smyček (control loops) – *„nekonečných smyček, které regulují stav systému“* (*Kubernetes Documentation*, ©2022, Controllers). Kontroléry (controllers) jsou řídicí smyčky, které sledují objekty a vykonávají změny, které přibližují aktuální stav objektů blíže požadovanému. Příkladem je situace, kdy při pádu uzlu klesne počet požadovaných replik aplikace, kontrolér tuto informaci zjistí a snaží se vyvolat jejich spuštění na jiném uzlu. (*Kubernetes Documentation*, ©2022, Controllers)

Do Kubernetes lze přidat vlastní typy objektů a kontroléry pomocí rozhraní CustomResourceDefinitions (CRD) (*Kubernetes Documentation*, ©2022, Custom Resources), čehož využívá například již zmiňovaný nástroj Rook sloužící k nasazení Cephu do Kubernetes, viz kapitola 1.3.5.

1.2.2 Architektura Kubernetes



Obrázek 1.1: Architektura Kubernetes
Zdroj: autor podle (*Kubernetes Documentation*, ©2022, Kubernetes Components; Spaleta, 2019)

Na obrázku 1.1 je znázorněna architektura Kubernetes s jeho základními součástmi. Pro lepší znázornění architektury jsou komponenty řídicí vrstvy na obrázku odděleny, ve skutečnosti jsou ale také umístěny na uzlech Kubernetes clusteru. S výjimkou kubelet (viz dále) jsou komponenty nasazeny jako pody (experimentálně zjištěno výpisem podů pomocí kubectl v testovacím clusteru). Obrázek také pro přehlednost zanedbává fakt, že některé komponenty mohou mít více instancí. Následující popis vyobrazených komponent vychází z dokumentace (*Kubernetes Documentation*, ©2022, Kubernetes Components).

Komponenty umístěné na každém uzlu:

- Kubelet: Spravuje kontejnery, které tvoří pody.
- Kube-proxy: Nastavuje síťová pravidla na uzlech upravující přístup k podům.
- Běhové prostředí pro kontejnery (např. containerd nebo CRI-O).

Komponenty řídicí vrstvy:

- Etcd: Silně konzistentní, vysoce dostupná distribuovaná databáze (*etcd*, ©2013–2022). V Kubernetes je využita k ukládání objektů popisujících cluster.
- Správce kontrolérů: V rámci něj běží kontroléry.

- Plánovač: Rozhoduje o umístění podů.
- API server: Poskytuje přístup ke Kubernetes API pro ovládání clusteru a čtení jeho vlastností. K tomuto API se připojuje i nástroj kubectl.

1.2.3 Nasazení aplikací do Kubernetes

Definice objektů, tedy zejména jejich požadovaného stavu, lze zapsat do souborů ve formátu YAML a poté nasadit do Kubernetes pomocí nástroje kubectl. Pomocí tohoto nástroje lze rovněž nasazené objekty číst a upravovat. (*Kubernetes Documentation*, ©2022, Understanding Kubernetes Objects)

Pro nasazení aplikací lze také využít nástroj Helm, který umožňuje nasazovat předpřipravené verzované sady objektů zvané „charts“, které jsou veřejně dostupné. (*Helm Docs*, ©2022, Docs, Charts)

Dále jsou přiblíženy vybrané typy objektů využívané pro nasazení aplikací a jejich konfiguraci; vzhledem k rozsahu Kubernetes se však jedná pouze o hrubý a nekompletní přehled.

Uživatel typicky nenasazuje pody přímo, místo toho jsou používány následující typy objektů, na základě kterých Kubernetes automaticky pody vytváří a spravuje:

- Deployment: V clusteru udržuje stanovený počet replik podu s danou konfigurací.
- StatefulSet: Je podobný typu Deployment, každá replika má ale unikátní identitu, která se uchovává přes její zánik a opětovné nasazení, takže jí je např. vždy přiděleno stejné úložiště.
- DaemonSet: Je podobný typu Deployment, místo pevně stanoveného počtu replik je ale umístěna jedna replika na každý uzel vyhovující požadavkům.
- Job: Specifikuje nasazení podů vykonávajících dočasné úlohy.

(*Kubernetes Documentation*, ©2022, Workloads, StatefulSets)

Součástí objektů těchto typů je definice šablony podu, podle které jsou pody vytvářeny. Šablona specifikuje mimo jiné konfiguraci kontejnerů, které pod tvoří: použitý obraz, vstupní bod, zveřejněné porty, připojená úložiště apod. Lze v rámci ní také konfigurovat, jakým způsobem Kubernetes ověřuje, jestli je pod „živý“, a jestli má být pod v případě pádu restartován. (*Kubernetes Documentation*, ©2022, Pods, Pod Lifecycle)

Pody lze nastavením jejich afinity k uzlům (node affinity) omezit na uzly s určitým štítkem (*Kubernetes Documentation*, ©2022, Assigning Pods to Nodes). Toho lze využít např. k tomu, aby byly pody vyžadující lokální úložiště umístěny na uzly s vhodnými disky. Jiným způsobem ovlivnění umístění podů je označení uzlů pomocí tzv. taint; toto označení brání nasazení podů, které nemají nastavenou potřebnou toleranci (toleration) (*Kubernetes Documentation*, ©2022, Taints and Tolerations).

Pro doplnění konfigurací aplikacím např. ve formě souborů pro čtení připojených ke kontejneru nebo proměnných prostředí slouží objekty typů ConfigMap a Secret.

ConfigMap typicky obsahuje běžnou konfiguraci, zatímco Secret je určen k oddělení citlivých informací. (*Kubernetes Documentation*, © 2022, ConfigMaps, Secrets)

Pomocí objektu typu služba (Service) lze aplikaci nasazené v Kubernetes přidělit pevnou DNS adresu. Kubernetes se stará o vyrovnávání zátěže mezi replikami podů, které aplikaci tvoří. (*Kubernetes Documentation*, © 2022, Service)

Externí úložiště lze připojit k podům pomocí abstrakce zvané „volume“, která umožňuje připojit úložiště různých typů, včetně např. CephFS (viz 1.3.3). (*Kubernetes Documentation*, © 2022, Volumes)

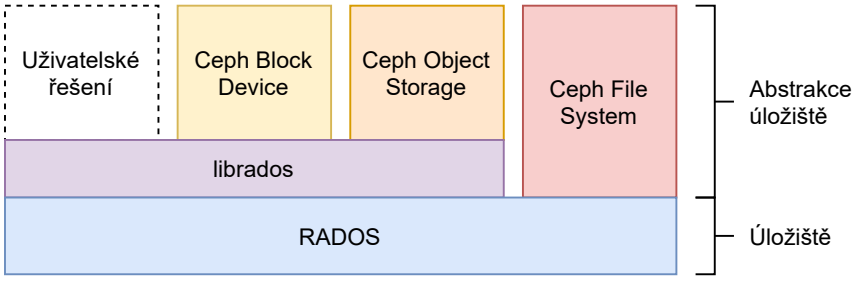
Pomocí rozhraní PersistentVolumeClaim lze vyžádat dynamické přidělení úložiště, kdy není určeno jeho konkrétní umístění, ale jen jeho požadované vlastnosti, a to prostřednictvím tzv. třídy úložiště. Třídy úložiště používané v clusteru lze definovat pomocí objektů typu StorageClass, které specifikují plugin poskytující úložiště (provisioner) a jeho parametry. (*Kubernetes Documentation*, © 2022, Dynamic Volume Provisioning, StorageClasses)

1.3 Ceph: distribuované úložiště

Ceph je distribuované, softwarově definované úložiště. Spoluautor Cephu Weil (2019, CEPH IS RELIABLE) ho popisuje jako *„spolehlivé úložiště z nespolehlivých komponent“*. Úložný prostor je poskytován přes několik typů rozhraní.

1.3.1 Architektura Cephu

Následující popis architektury úložiště Ceph vychází z dokumentace (*Ceph Documentation*, © 2016 [b.r.], Architecture, Ceph Glossary).



Obrázek 1.2: Logická architektura Cephu
Zdroj: autor podle (*Ceph Documentation*, © 2016 [b.r.], Architecture)

Logická architektura Cephu je znázorněna na obrázku 1.2. Základem Cephu je distribuované úložiště RADOS, které ukládá obecné bloky dat (viz 1.3.2). Na RADOS jsou postaveny vrstvy realizující rozhraní různých typů úložišť:

- Souborový systém (Ceph File System, zkráceně CephFS): Viz 1.3.3.
- Objektové úložiště (Ceph Object Storage): Viz 1.3.4.

- Blokové zařízení (Ceph Block Device): V systému pro přepis není používáno a z tohoto důvodu není dále popisováno.

Librados je knihovna, která umožňuje vývojářům přímo přistupovat k úložišti RADOS.

Cluster Cephu je tvořen následujícími démony:

- Monitor (MON) a Object Storage Daemon (OSD): Tvoří cluster úložiště RADOS, viz kapitola 1.3.2.
- Manager (MGR): V rámci něj běží moduly sledující stav Ceph clusteru nebo poskytující tyto informace uživateli a externím systémům. Jedním z modulů je webový přehled (dashboard). (*Ceph Documentation*, ©2016 [b.r.], Ceph Manager Daemon, Ceph Dashboard)
- Metadata Server (MDS): Slouží ke správě metadat v CephFS, viz kapitola 1.3.3.

1.3.2 RADOS

RADOS (Reliable Autonomic Distributed Object Store) je úložiště, které je jádrem Cephu. Je silně konzistentní, škálovatelné, vysoce dostupné a poskytuje ochranu před ztrátou dat. (Weil, 2019, CEPH IS A UNIFIED STORAGE SYSTEM)

RADOS cluster, znázorněný na obrázku 1.3, tvoří následující démoni:

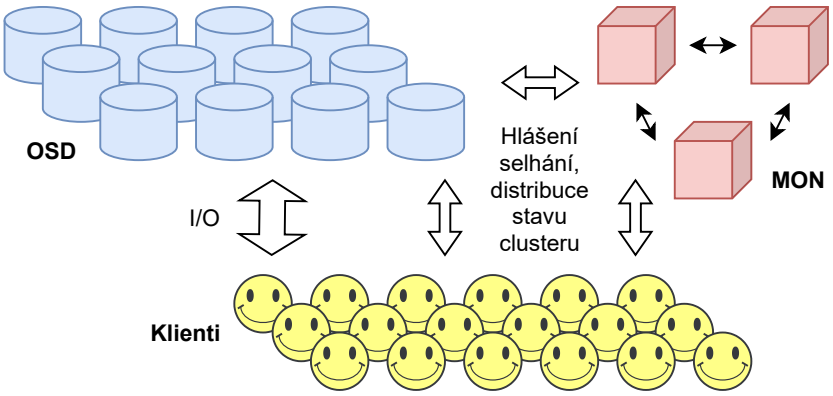
- Object Storage Daemon (OSD): Spravuje jeden svěřený úložný prostor (typicky diskový oddíl). OSD vzájemně spolupracují na replikaci a rovnoměrném rozmístění dat. RADOS podporuje i tisíce OSD v jednom clusteru.
- Monitor (MON): Udržují hlavní kopii (master copy) tzv. mapy clusteru (cluster map), viz dále. V clusteru se typicky nachází 3–7 MON.

(Weil et al., 2007, s. 2; Weil, 2019, RADOS SOFTWARE COMPONENTS)

RADOS ukládá data v podobě tzv. objektů (object), jejichž velikost je ve výchozím nastavení 4 MB. Každý objekt má přiřazený tzv. pool, do kterého patří. Pooly jsou definované uživatelem a slouží k logickému seskupení objektů, na něž mají být aplikována podobná pravidla pro ukládání; nejedná se o fyzické seskupení. (Weil, 2019, RADOS DATA OBJECTS)

Pro určení rozmístění dat není používán index, ale poloha dat je počítána na základě zmíněné mapy clusteru, která obsahuje informace o topologii a stavu RADOS clusteru. Výpočet provádí sami klienti a OSD, kteří u sebe kopii mapy clusteru mají. Revize mapy clusteru se mezi jejími držiteli distribuují při vzájemné komunikaci. (Weil et al., 2007, s. 2, 3)

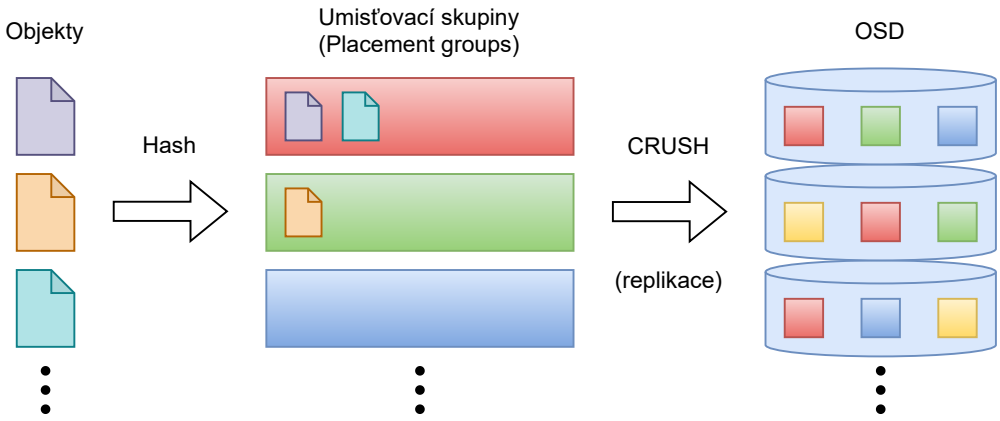
Obrázek 1.4 zobrazuje princip rozmísťování objektů na OSD popisovaný dále. Každý pool má nastavené určité množství tzv. umisťovacích skupin (placement



Obrázek 1.3: RADOS cluster
Zdroj: autor, adaptováno z (Weil et al., 2007, s. 2)

groups). Každý objekt je pomocí hašovací funkce aplikované na jeho název zařazen do jedné z umisťovacích skupin svého příslušného poolu. (Weil et al., 2007, s. 2; Weil, 2019, RADOS DATA OBJECTS)

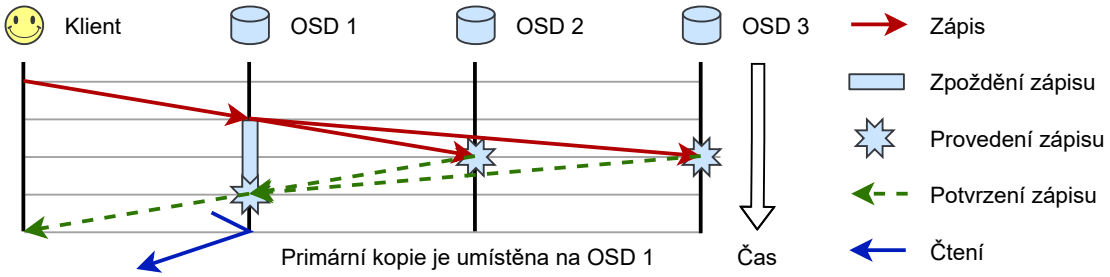
Umisťovací skupiny jsou ukládány na OSD jako celek. Na OSD jsou rozmisťovány pomocí algoritmu CRUSH (Weil et al., 2006), který jako vstup přijímá mapu clusteru a název umisťovací skupiny. CRUSH je autory svým účelem přirovnáván k hašovací funkci – výsledné rozmístění je rovnoměrné –, jeho důležitou vlastností ale je, že při změně topologie clusteru mění umístění pouze nutného množství dat. Na jednom OSD se mohou nacházet umisťovací skupiny z různých poolů. (Weil et al., 2007, s. 2–3; Weil, 2019, RADOS DATA OBJECTS, PLACING PGS WITH CRUSH)



Obrázek 1.4: Umístění objektů na OSD s replikací
Zdroj: autor podle (Weil, 2019, čas 17:00)

Pro ochranu dat lze využít několik technik pro redundantní ukládání, v systému pro přepis je použita prostá replikace, kdy jsou na OSD ukládány kopie umisťovacích skupin jako celek. Pro replikaci používá Ceph techniku primární kopie (primary copy). Zapisování a čtení objektů za použití této techniky je zobrazeno na obrázku 1.5. Jejím důsledkem je, že přestože úložiště obsahuje více replik objektu,

čtena je vždy pouze primární. Původní článek o RADOS popisuje i další techniky replikace, ty se ale v dokumentaci Cephu nevyskytují, neboť již nejsou podporovány. (Weil et al., 2007, s. 4; Weil, 2015; *Ceph Documentation*, ©2016 [b.r.], Architecture)



Obrázek 1.5: Replikace technikou „primární kopie“
Zdroj: autor, adaptováno z (Weil et al., 2007, s. 4)

O umisťovacích skupinách Weil (2019, WHY PLACEMENT GROUPS?) zmiňuje, že byly zvoleny jako kompromis mezi dvěma extrémy: replikací celých disků a replikací každého objektu zvlášť. Pokud by byly replikovány celé disky, nemohl by při selhání disku být paralelizován přesun dat na náhradní disk, zatímco při použití umisťovacích skupin mohou být jejich náhradní repliky rovnoměrně rozmístěny přes cluster, čímž se přesun dat paralelizuje. Na druhou stranu, pokud by byl replikován každý objekt zvlášť, dramaticky by narostla pravděpodobnost, že by při vícenásobném selhání disků došlo ke ztrátě všech replik alespoň jednoho objektu.

1.3.3 Ceph File System: souborový systém

Ceph File System (CephFS) je silně konzistentní POSIX-kompatibilní souborový systém se souběžným sdíleným přístupem. (*Ceph Documentation*, ©2016 [b.r.], Ceph File System; Weil, 2019, CEPHFS: CEPH FILE SYSTEM)

Soubory a metadata jsou uloženy v RADOS v oddělených poolech. Soubory jsou rozděleny do RADOS objektů. K souborům přistupuje klient přímo, zatímco přístup k metadatům zprostředkovává komponenta Metadata Server (MDS). (Weil, 2019, CEPHFS: CEPH FILE SYSTEM)

MDS mimo jiné spravují adresářový strom a metadata souborů a koordinují přístup klientů k souborům. MDS se v clusteru mohou nacházet jednotky až desítky. Každý MDS má na starosti část adresářového stromu, který je mezi ně dynamicky rozdělen podle míry zátěže (dynamic subtree partitioning). (Weil, 2019, CEPH-MDS: CEPH METADATA SERVER)

1.3.4 Ceph Object Storage: objektové úložiště

Ceph Object Storage umožňuje pracovat s daty pomocí rozhraní S3 a Swift pro tzv. objektové úložiště (*Ceph Documentation*, ©2016 [b.r.], Ceph Object Gateway). Rozhraní S3 je původem rozhraním cloudové služby Amazon S3 (*Amazon S3*, ©2022)

poskytující objektové úložiště, toto rozhraní ale začala používat i řada jiných implementací objektových úložišť. Druhé podporované rozhraní, Swift, není v systému pro přepis používáno.

Klienti využívající objektové úložiště se přes jmenovaná rozhraní připojují k Ceph Object Gateway (také označována RADOS Gateway), jež je samostatnou komponentou komunikující s RADOS clusterem. Objekty jsou ukládány do RADOS poolu, kde jsou rozděleny do RADOS objektů[3]. (*Ceph Documentation*, ©2016 [b.r.], Ceph Object Gateway; Weil, 2019, RGW: RADOS GATEWAY)

1.3.5 Rook: nástroj pro nasazení Cephu do Kubernetes

Rook je open-source nástroj zaštiťovaný CNCF, pomocí kterého lze nasadit Ceph do Kubernetes a automatizovat jeho správu. Rook přidává do Kubernetes vlastní typy objektů, které slouží k popisu Ceph clusteru, jednotlivých úložišť a dalších konceptů. Program Rook Operator, který je nasazený v Kubernetes clusteru, na základě těchto objektů nasazuje Ceph cluster do Kubernetes a spravuje ho. (*Rook Ceph Documentation*, ©2022, Rook, Ceph Quickstart)

1.4 Prometheus: sběr a ukládání metrik

Prometheus je open-source nástroj pro sběr a ukládání metrik zastřešovaný CNCF. (*Prometheus Docs*, ©2014–2022, Overview)

Metriky (metrics) jsou naměřené hodnoty veličin, například:

- aktuální objem paměti využívané kontejnerem;
- maximální objem paměti, který může kontejner využít, nastavený v Kubernetes;
- celkový objem dat odeslaných kontejnerem po síti od jeho spuštění.

Metriky jsou zveřejňovány komponentami systému, viz kapitola 1.4.2. V rámci monitorování systému jsou shromažďovány, aby mohla být odhalena a diagnostikována nežádoucí chování.

1.4.1 Datový model Promethea

Prometheus uvažuje metriky v podobě tzv. časových řad (time series) – chronologicky uspořádaných sledů naměřených hodnot s příslušnými časovými značkami. Časová řada je v Prometheu určena názvem metriky a unikátní kombinací štítků (labels). Štítek se skládá z názvu štítku a hodnoty štítku, přičemž v popisu jedné

[3]Objekty objektového úložiště a RADOS objekty nejsou totéž.

časové řady se názvy štítků neopakují. Unikátními časovými řadami jsou dle uvedených pravidel např. následující (notace dle dokumentace):

- `nazev_metriky{stitek_a="X"}`
- `nazev_metriky{stitek_a="X", stitek_b="Y"}`
- `nazev_metriky{stitek_a="Y", stitek_b="Y"}`
- `nazev_jine_metriky{stitek_a="Y", stitek_b="Y"}`

(*Prometheus Docs*, ©2014–2022, Data model)

Prometheus k sebraným metrikám z každého zdroje (viz 1.4.2) automaticky přidává následující štítky:

- job (úloha): Obsahuje název úlohy, který je nastavený v Prometheu. V praxi to bývá například název aplikace nebo komponenty.
- instance: Obsahuje adresu hostitele a port, které přísluší dané instanci úlohy.

(*Prometheus Docs*, ©2014–2022, Jobs and instances)

Další štítky přiřazuje konkrétní zdroj metrik nebo je nastavuje Prometheus na základě konfigurace zdroje. (*Prometheus Docs*, ©2014–2022, Exposition formats, Getting Started)

Při používání štítků je nutné dávat pozor na kardinalitu – množství hodnot, kterých může štítek nabývat. Každá unikátní kombinace štítků vytváří novou časovou řadu; štítky s vysokou kardinalitou a zvláště pak jejich kombinace ústí ve velké množství časových řad, což zvyšuje zátěž na systém. Problém se netýká štítků, jejichž hodnota vždy závisí na kombinaci ostatních štítků a mají charakter doplňkové informace. (*Prometheus Docs*, ©2014–2022, Metric and label naming; Marchbanks, 2019)

Metriky mohou být typu counter, gauge, histogram, nebo summary. V rámci práce se pracovalo s prvními dvěma zmíněnými:

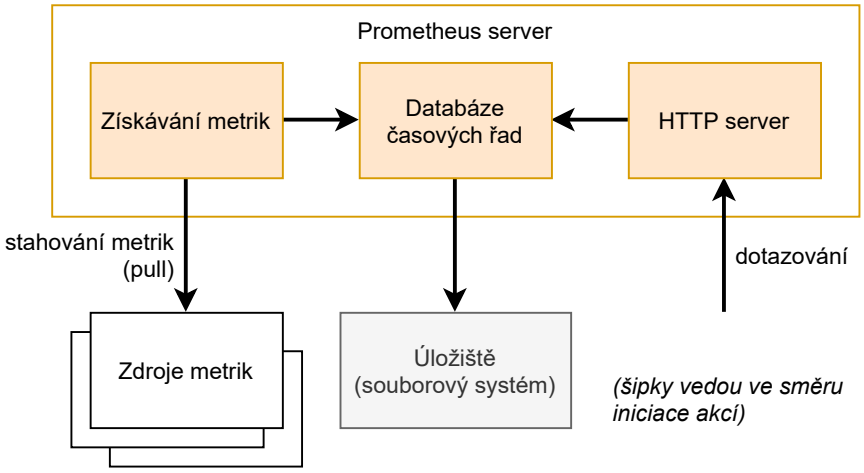
- Counter: Monotónní inkrementální veličina s počátkem v nule.
- Gauge: Veličina, která v čase nabývá libovolné hodnoty.

(*Prometheus Docs*, ©2014–2022, Metric types)

U typu counter je předpokládáno, že pokud je následující hodnota menší než předcházející, nastalo vynulování čítače, např. z důvodu pádu zdroje. Tohoto předpokladu využívá dotazovací jazyk Promethea (viz 1.4.3): funkce pracující s metrikami typu counter hodnoty uvažují tak, jakoby k vynulování nedošlo. (*Prometheus Docs*, ©2014–2022, Query functions)

1.4.2 Architektura Promethea, sběr a ukládání metrik

Vybrané části architektury Promethea jsou znázorněny na obrázku 1.6. Základní komponentou Promethea je Prometheus server, jehož základní činností je získávání metrik ze zdrojů a jejich ukládání do databáze časových řad. V systému pro přepis je nasazena jedna instance Prometheus serveru. (*Prometheus Docs*, ©2014–2022, Overview)



Obrázek 1.6: Architektura Promethea (vybrané části)
Zdroj: autor podle (*Prometheus Docs*, ©2014–2022, Overview)

Preferovaným a nejčastěji využívaným způsobem získávání metrik pomocí Promethea je jejich stahování serverem ze zdrojů (tzv. „pull“ přístup, scraping). Každý zdroj metrik poskytuje HTTP koncový bod, na kterém jsou zveřejněny aktuální hodnoty metrik (empiricky zjištěno v rámci rešerše). Pro tvorbu těchto koncových bodů jsou dostupné knihovny. Metriky jsou z koncových bodů serverem periodicky stahovány; v systému pro přepis je perioda nastavena na 15 sekund. Ze způsobu zveřejňování metrik je zřejmé, že pokud zdroj aktualizuje metriky dříve, než je server stáhne, jsou předešlé hodnoty ztraceny. (*Prometheus Docs*, ©2014–2022, Getting started, FAQ, Client libraries)

Pro ukládání metrik používá Prometheus ve výchozím nastavení tzv. lokální úložiště (local storage) – adresář v POSIX-kompatibilním souborovém systému. V systému pro přepis jsou data ukládána do vyhrazeného CephFS poolu. (*Prometheus Docs*, ©2014–2022, Storage)

1.4.3 Dotazování Promethea

Prometheus poskytuje HTTP API pro čtení metrik a dalších informací. Pro dotazování Promethea je používán jeho vlastní jazyk zvaný PromQL, který mimo jiné umožňuje:

- výběr časových řad s určitým názvem metriky a hodnotami štítků, kde jsou hodnoty štítků omezeny přímým porovnáním nebo regulárním výrazem;

- výběr hodnot k jednomu časovému okamžiku;
- výběr vektoru hodnot v čase;
- agregace napříč časovými řadami pro jeden časový okamžik;
- agregace v čase v rámci jedné časové řady;
- binární operace s vektory metrik.

(*Prometheus Docs*, ©2014–2022, sekce Querying)

Pomocí tzv. „recording rules“ lze v pravidelných intervalech vypočítávat a ukládat výsledky dotazů. Uložené výsledky jsou potom dostupné také jako časová řada. (*Prometheus Docs*, ©2014–2022, Recording rules)

Příklad konkrétního využití HTTP API a dotazování pomocí PromQL je uveden v kapitole 2.3.

1.5 Grafana Loki: sběr a ukládání logů

Grafana Loki je open-source nástroj pro sběr a ukládání logů vyvíjený společností Grafana Labs. Oficiálně je popisován jako *„Prometheus pro logy“*. (*Grafana Loki documentation*, ©2022a)

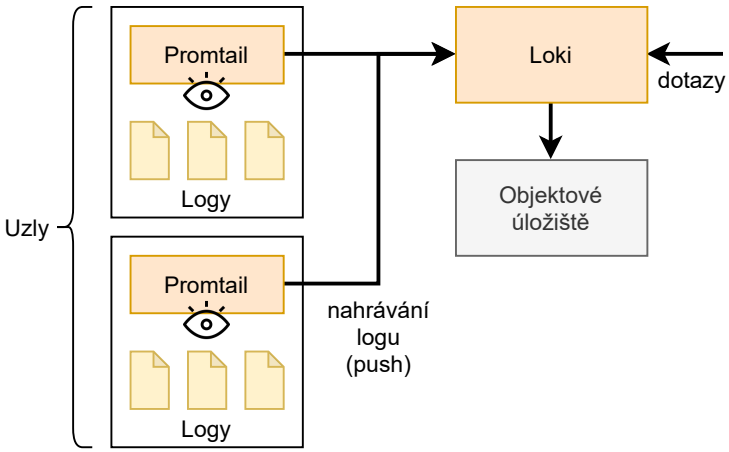
1.5.1 Datový model Lokiho

Loki používá podobný datový model jako Prometheus (viz 1.4.1). Sesbírané řádky logů s časovými značkami jsou tříděny do proudů (streams), jež jsou určeny unikátní kombinací štítků (labels), které fungují stejně jako u Promethea. Na rozdíl od Promethea ale Loki nemá ekvivalent názvu metriky, proudy jsou určeny pouze štítky. Obdobně jako Prometheus přiřazuje Loki proudům štítky „job“ a „instance“. Další štítky mohou být zdroji přiřazeny u jeho definice v Lokim. Stejně jako u Promethea je třeba dávat pozor na kardinalitu štítků. (*Grafana Loki documentation*, ©2022a, Labels)

1.5.2 Architektura Lokiho, sběr a ukládání logů

Na obrázku 1.7 je znázorněno použití Lokiho v systému pro přepis. Loki umožňuje nasazení v podobě monolitické aplikace i v podobě mikroslužeb plnících dílčí funkce (*Grafana Loki documentation*, ©2022a, Architecture); v systému pro přepis je nasazena jedna monolitická instance Lokiho.

Loki používá ke sběru logů agenty zvané „klienti“ (clients), kteří získávají logy a zasílají mu je (tzv. „push“ přístup). V systému pro přepis je používán klient Promtail, který umožňuje automaticky sbírat logy ze všech podů na uzlu Kubernetes clusteru, na kterém je nasazen. Promtail sbírá log průběžně; časové značky logu jsou ve výchozí konfiguraci nastaveny podle času sběru. (*Grafana Loki documentation*, ©2022a, Overview, Clients, Promtail, Promtail – timestamp stage)



Obrázek 1.7: Vnější architektura Lokiho v systému pro přepis
Zdroj: autor

Loki umožňuje ukládat data do objektového úložiště s rozhraním S3. V systému pro přepis jsou data ukládána do vyhrazeného Ceph Object Storage poolu. (*Grafana Loki documentation*, © 2022a, Architecture)

Dle dokumentace se Loki od jiných řešení pro ukládání logu odlišuje tím, že indexuje pouze štítky přiřazené řádkám logu a nikoliv text logu. Filtrování řádek logu podle jejich obsahu je při dotazu prováděno ad hoc výpočtem, který je distribuovaný, pokud je Loki horizontálně škálován. Dokumentace uvádí, že toto řešení bylo zvoleno pro snížení zátěže. (*Grafana Loki documentation*, © 2022a, Overview, Architecture, LogQL)

1.5.3 Dotazování Lokiho

Obdobně jako Prometheus poskytuje Loki pro čtení uloženého logu a jiných informací HTTP API. Formát odpovědi API Lokiho je kompatibilní s formátem odpovědi API Promethea. Pro dotazování Lokiho je používán jeho vlastní jazyk LogQL, který mimo jiné umožňuje:

- výběr proudů logu na základě štítků,
- filtrování řádek logu podle jejich obsahu pomocí regulárních výrazů,
- vypočtení metrik z logů a provádění výpočtů nad nimi.

(*Grafana Loki documentation*, © 2022a, HTTP API, sekce LogQL)

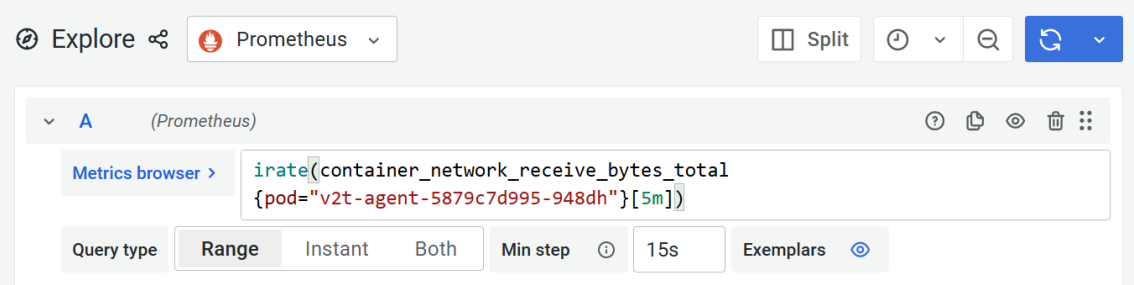
Příklady konkrétního použití HTTP API a dotazování pomocí LogQL jsou uvedeny v kapitole 2.4.

1.6 Grafana: vizualizace dat

Grafana je open-source nástroj pro vizualizaci dat vyvíjený společností Grafana Labs. Poskytuje webové rozhraní, ve kterém lze vytvářet a zobrazovat interaktivní

přehledy (dashboard), které obsahují především grafy dat. Nativně podporuje propojení s řadou datových zdrojů, mezi než patří i Prometheus a Loki. (*Grafana documentation*, © 2022, Introduction to Grafana, Dashboards, Data sources)

Součástí webového rozhraní Grafany je nástroj Explore (na obrázku 1.8), který umožňuje vizualizaci výsledků ad hoc dotazů na datové zdroje. Explore zároveň pomáhá s tvorbou dotazů: např. zobrazuje názvy dostupných metrik a štítků v Prometheu a podporuje automatické dokončování při psaní. (*Grafana documentation*, © 2022, Explore, Query management)



Obrázek 1.8: Ukázka rozhraní nástroje Grafana Explore (konfigurace dotazu)
Zdroj: autor

1.7 Jádro systému pro přepis a požadavky na přepis

Jádro systému pro přepis (dále také „jádro“) je komponenta, která má na starosti hlavní činnost systému: vykonávání požadavků uživatelů na přepis řeči na text (dále „požadavek na přepis“). Je naprogramováno v jazyce Go (viz 1.9). Podobně jako systém Kubernetes je jádro řízeno na principu objektů a řídících smyček (viz 1.2.1).

Vzhledem k rozsahu jádra jsou zmíněny pouze vybrané komponenty:

- Etcd: Stejně jako u Kubernetes je tato databáze použita k ukládání objektů (viz 1.2.2).
- Agent: V rámci něj běží tzv. služby (service; dále označované „služby pro přepis“), což jsou programy, které realizují přepis řeči na text.
- Brána: Přijímá uživatelský požadavek, zařizuje jeho vykonání a propojuje uživatele s vykonávajícím agentem.
- Plánovač: Vybírá agenta, který vykoná požadavek uživatele.

Pokud chce uživatel nechat přepsat nahrávku, je průběh zhruba následující (graficky znázorněno na obrázku 1.9):

1. Uživatel odešle požadavek na přepis na API jádra.
2. Plánovač určí, který agent má požadavek vykonat.

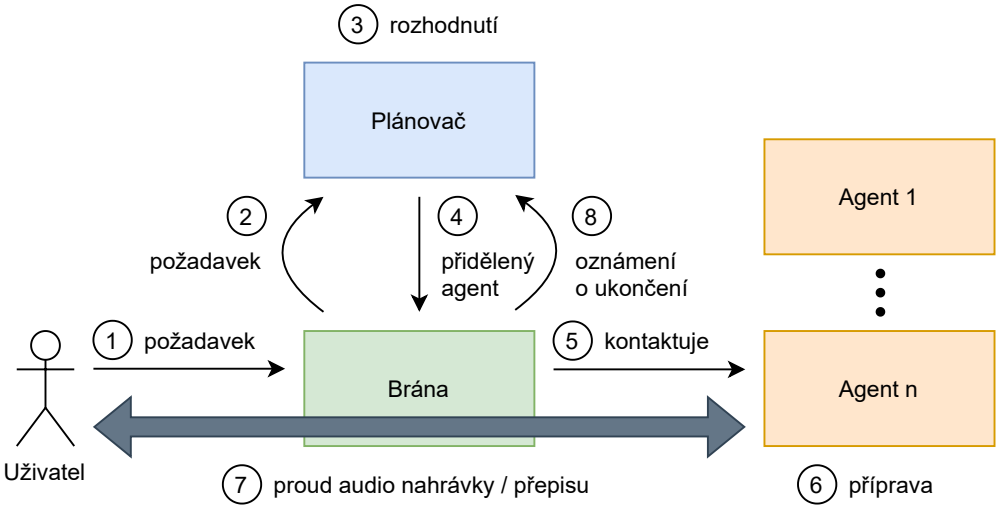
3. Agent se připraví na vykonání požadavku (např. stáhne z repozitáře systému potřebnou službu pro přepis, pokud ji nemá dostupnou lokálně).

4. Uživatel je propojen s agentem, agentovi vysílá (stream) audio nahrávku a agent mu zpět vysílá její přepis.

5. Po ukončení přepisu je o tomto plánovač informován.

V jádru systému pro přepis je každý požadavek na přepis reprezentován objektem uloženým v etcd (dále označován „objekt požadavku“), který ukládá jeho popis a informace o průběhu jeho vykonávání. Pro přístup k objektům je jádrem poskytováno HTTP API. K dispozici je také webový přehled, ve kterém lze po přihlášení zobrazit historii požadavků na přepis a informace o nich. Zadávání požadavků umožňuje utilita s rozhraním v příkazové řádce.

Délka trvání vykonání jednoho požadavku se dle vedoucího bakalářské práce pohybuje v praxi v řádech sekund až hodin, výjimečně dní.



Obrázek 1.9: Přibližný průběh požadavku na přepis
Zdroj: autor

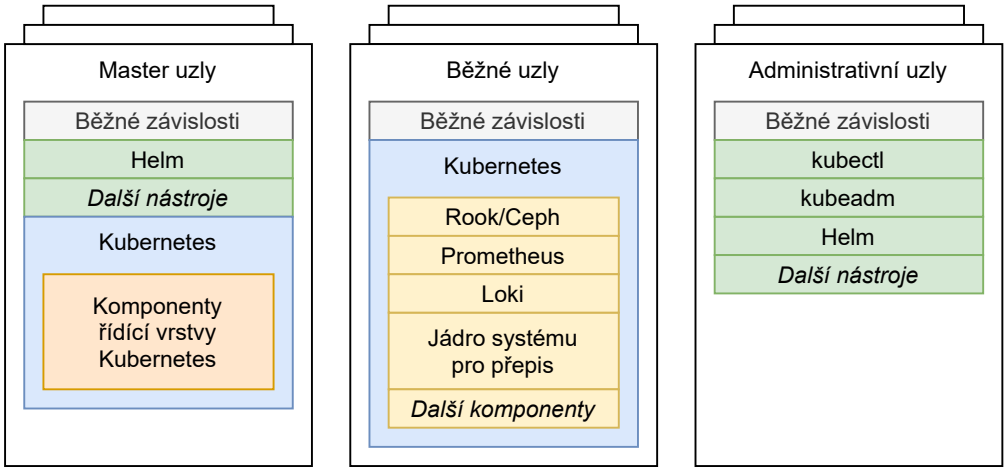
1.8 Cluster systému pro přepis a jeho nasazení

Tato kapitola doplňuje některé detaily týkající se clusteru systému pro přepis a způsobu, jakým je nasazován.

Jak je na obrázku 1.10 znázorněno, v clusteru systému pro přepis jsou rozlišeny tři typy uzlů podle nasazených komponent a nainstalovaných nástrojů:

- Master uzly: Jsou součástí Kubernetes clusteru. Běží na nich pouze komponenty řídící vrstvy Kubernetes a jiné komponenty související s řízením clusteru.

- Běžné uzly: Jsou součástí Kubernetes clusteru. Běží na nich většina komponent, které tvoří systém pro přepis.
- Administrativní uzly: Jsou na ně nainstalovány administrativní nástroje.



Obrázek 1.10: Rozdělení uzlů clusteru systému pro přepis
Zdroj: autor

Znázornění clusteru na obrázku 1.10 opomíjí některé detaily:

- Zobrazená komponenta může, ale nemusí být na konkrétním uzlu nasazena.
- Některé komponenty se ve skutečnosti skládají z více samostatných součástí.
- Některé komponenty nebyly pro zjednodušení zahrnuty.

K vytvoření prvního uzlu Kubernetes clusteru a začleňování uzlů do něj slouží nástroj kubeadm (*Kubernetes Documentation*, ©2022, Creating a cluster with kubeadm). Řada komponent je do Kubernetes clusteru instalována pomocí správce balíčků Helm (viz 1.2.3).

Pro automatizaci procesu nasazování systému pro přepis je jeho autory vyvíjeno řešení využívající nástroj Ansible, které je blíže popsáno v kapitole 3.2.

1.9 Programovací jazyk Go

Go je programovací jazyk zaštiťovaný společností Google, který je široce používán v oblasti distribuovaných systémů. Je v něm implementováno velké množství technologií zmiňovaných v rámci práce, např. Kubernetes („kubernetes/kubernetes“, 2021), Prometheus (*Prometheus Docs*, ©2014–2022, Overview) nebo Loki („grafana/loki“, 2022).

Jedná se o kompilovaný, staticky typovaný jazyk s garbage collectorem. Podporuje objektově orientované programování a součástí jazyka jsou mechanismy zjednodušující programování souběžných úloh. Syntaxe vychází z jazyka C. (*Documentation - The Go Programming Language*, [b.r.])

2 Vývoj aplikace pro stahování diagnostických dat

Jedním z cílů bakalářské práce bylo vyvinutí aplikace, která ze systému pro přepis stáhne diagnostická data o vybraném požadavku na přepis a uloží je. Účelem je poskytnout možnost diagnostikovat na základě stažených dat problémy s během požadavků, a to buď samotnými uživateli, nebo jinou osobu, jíž jsou data zaslána. Výstup aplikace má zahrnovat informace o požadavku na přepis a relevantní log a metriky z agenta, který požadavek vykonával. Je požadováno, aby byl výstup aplikace zapsán ve formátu vhodném pro analýzu člověkem; výstup aplikace nemá sloužit ke strojovému zpracování.

Následující podkapitoly popisují vývoj aplikace počínaje návrhem řešení, přes návrh a implementaci dílčích částí aplikace, až po její testování. Aplikace byla v průběhu vývoje průběžně konzultována s vedoucím bakalářské práce, aby odpovídala požadavkům.

2.1 Návrh

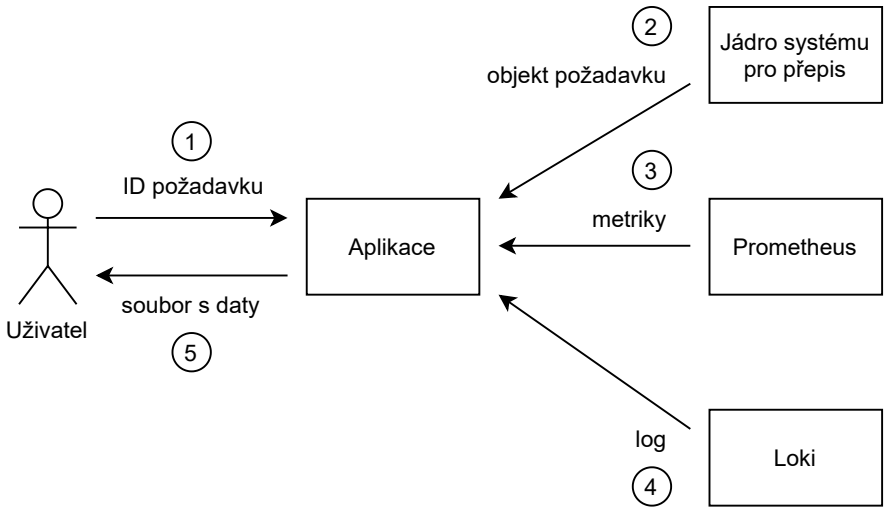
2.1.1 Data, jejich zdroje a obecný návrh řešení

Obrázek 2.1 zobrazuje obecný návrh řešení, který vychází z dostupných dat a jejich zdrojů popsaných dále.

Informace o požadavku na přepis jsou obsaženy v objektu požadavku, který lze získat z API jádra systému pro přepis. Objekt požadavku je také nutný pro stažení metrik a logu, protože je k jejich získání nejprve potřeba zjistit, který agent v jakém časovém období požadavek vykonával. Požadavky jsou jednoznačně identifikovány svým názvem (náhodné ID), který tak může sloužit pro výběr požadavku uživatelem.

Metriky z agenta lze získat dotazováním nástroje Prometheus. Vedoucím bakalářské práce bylo upřesněno, že výstupní informace získané z metrik mají být podobné informacím dostupným v Grafaně systému pro přepis v přehledu „Kubernetes / Compute Resources / Pod“ (dále označován „vzorový přehled“), který pochází z chart pro nástroj Helm (viz 1.2.3) s názvem kube-prometheus-stack („prometheus-community/helm-charts“, 2021, charts/kube-prometheus-stack).

Log z agenta lze získat dotazováním nástroje Loki. Počet řádek logu může být libovolný, takže je třeba brát v úvahu i případné velké objemy dat.



Obrázek 2.1: Návrh aplikace pro stahování diagnostických dat
Zdroj: autor

2.1.2 Způsob realizace

Původním předpokladem, který byl nakonec realizován, bylo, že pro stahování diagnostických dat bude vyvinuta vlastní aplikace. Zvoleným programovacím jazykem bylo Go, aby mohly být použity knihovny systému pro přepis a kvůli jednotnosti se zbytkem projektu.

Na počátku byla zvažována i varianta řešení ve formě přehledu v Grafaně. Využití jejích možností pro získávání dat ze zdrojů a jejich vizualizaci by mohlo zjednodušit implementaci, a navíc by zobrazení dat o požadavcích bylo na jednom místě s dalšími přehledy pro diagnostiku. Na základě konzultace s vedoucím bakalářské práce ale byla tato potenciální varianta řešení opuštěna, neboť zobrazení přehledu v Grafaně vyžaduje přístup k danému nasazení systému pro přepis, zatímco je požadováno, aby řešení data stahovalo a mohla být zaslána někomu, kdo tento přístup nemá. Nebyla tak dále zkoumána proveditelnost tohoto řešení.

2.1.3 Výstup aplikace

Výstup aplikace byl stanoven na základě konzultace s vedoucím bakalářské práce, aby jeho obsah odpovídal požadavkům. Bylo rozhodnuto, že výstup bude obsahovat:

- grafy metrik z agenta, na kterém běžel požadavek, určené pro vizuální analýzu;
- kompletní log z agenta, na kterém běžel požadavek, tak, aby bylo možné snadno vyhledávat v celém jeho objemu;
- informace obsažené v objektu požadavku.

Grafy metrik byly navrženy na základě vzorového přehledu v Grafaně, viz kapitola 2.1.1. Stanoveny byly následující:

- CPU:
  - Využití: Využití procesoru v porovnání s vyžádanou kapacitou a limitem využití nastavenými v Kubernetes.
  - Throttling.
- Paměť:
  - Využití: Velikost pracovní sady (working set) v porovnání s vyžádaným objemem paměti a limitem využití paměti nastavenými v Kubernetes.
- Síť:
  - Využití: Velikost toku příchozích a odchozích dat.
  - Příchozí pakety: Míra příchozích a příchozích-ztracených paketů.
  - Odchozí pakety: Míra odchozích a odchozích-ztracených paketů.

Bylo zvažováno, že by metriky mohly být obsaženy ve výstupu také v podobě surových dat. Na základě konzultace bylo toto shledáno nepotřebným, neboť jevy, které si klade analýza metrik za cíl odhalit – např. nízká velikost toku příchozích dat v celém průběhu požadavku –, jsou člověku patrné z grafu.

2.1.4 Uživatelské rozhraní

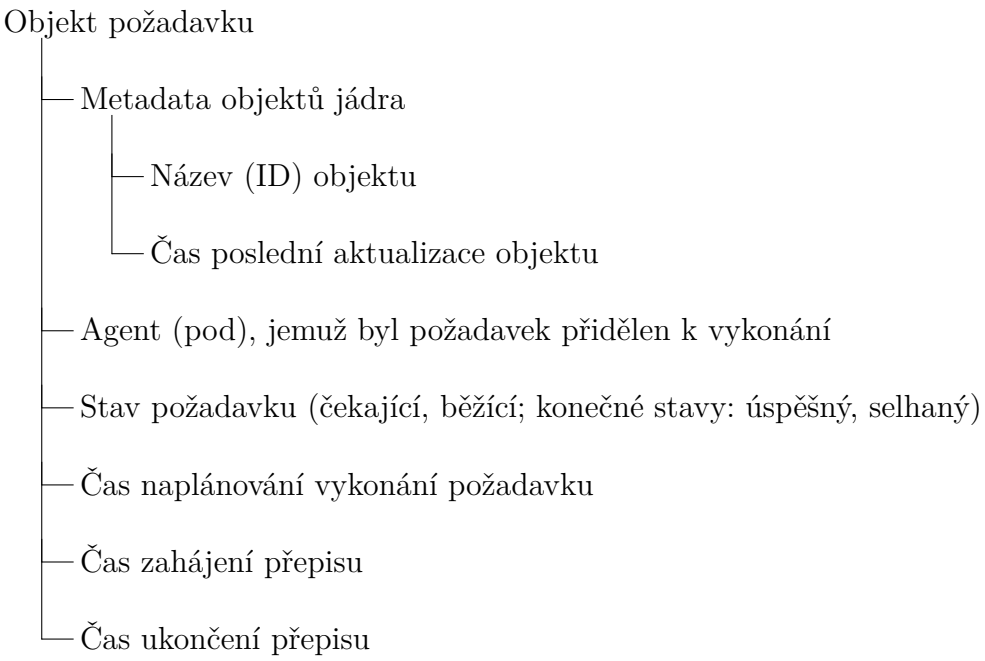
Z návrhu aplikace vyplývá, že je interakce uživatele s aplikací omezena na předání ID požadavku a případných doplňujících parametrů ovlivňujících chování aplikace. K ovládání aplikace tak postačuje rozhraní příkazové řádky, které je jednodušší na implementaci než grafické rozhraní a které bylo v době vývoje používáno i jinými programy pro práci se systémem pro přepis.

Mezi požadavky na aplikaci nebylo vytvoření finálního rozhraní určeného pro uživatele. Nebylo tak např. zvažováno, jestli je příkazová řádka pro budoucí uživatele dostatečně přívětivá. Rozhraní mělo sloužit k ovládání programu v současné fázi vývoje systému pro přepis a nebylo známo, jestli zůstane konečným.

2.2 Stažení a interpretace objektu požadavku

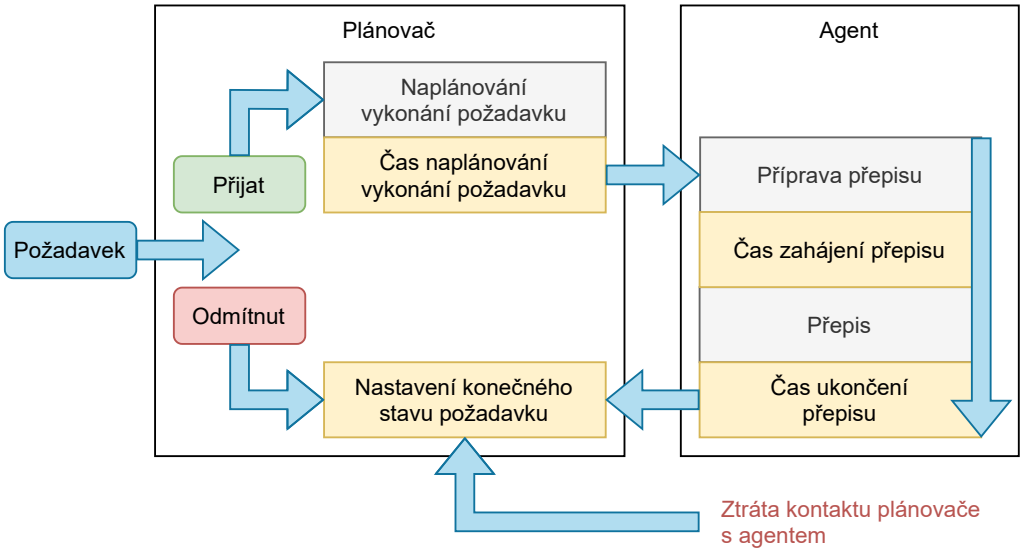
Objekt požadavku je stahován z API systému pro přepis, které vyžaduje autentizaci a objekty zasílá v serializovaném formátu Protocol Buffers verze 3 (*Protocol Buffers*, [b.r.]). Pro komunikaci s jádrem a převod objektu do struktur jazyka Go je používán klient, který je součástí kódu jádra systému pro přepis.

Objekt požadavku obsahuje řadu údajů; na obrázku 2.2 jsou uvedeny ty, které jsou v rámci práce zmiňovány.



Obrázek 2.2: Struktura objektu požadavku (vybraná pole)
Zdroj: autor

Časové značky mohou mít hodnotu značící, že časová značka nebyla nastavena. Kvůli rozchodu hodin mezi stroji v clusteru je vhodné znát, kterou komponentou jsou jednotlivé časy určovány, protože různé časy lze v různých kontextech považovat pouze za přibližné. Kdo a kdy určuje které časy a stavy za kterých situací je znázorněno na obrázku 2.3.



Obrázek 2.3: Průběh požadavku na přepis s původci časových značek a konečného stavu v objektu požadavku

Zdroj: autor

Za účelem stanovení základních hranic časového období, ze kterého jsou staženy metriky a log, byla zvolena následující interpretace:

- Počátkem časového období je čas naplánování vykonání požadavku: tento čas udává, od kdy nejdříve se začal agent požadavkem zabývat.
- Pokud se požadavek nachází v jednom z konečných stavů, je na základě konzultace s vedoucím bakalářské práce za konec časového období považován čas poslední aktualizace objektu. Je předpokládáno, že pokud je požadavek v konečném stavu, nedochází již k úpravám objektu požadavku, které by tuto časovou značku aktualizovaly. Bylo by sice možné využít čas ukončení přepisu, ten ale není spolehlivý, neboť není nastaven v případě ztráty kontaktu plánovače s agentem.
- Pokud se požadavek ještě nenachází v konečném stavu, ale již bylo naplánováno jeho vykonání, je jako konec časového období použit aktuální čas.
- Pokud by byly v objektu požadavku z důvodu chyby špatně vyplněné časy, mohlo by to vyústit v dotazy na metriky a log z rozsáhlého časového období. Proti tomu byla zvolena ochrana v podobě maximální délky časového období, ze kterého jsou data stažena. Na základě konzultace s vedoucím bakalářské práce byla stanovena na 7 dní s tím, že požadavky této délky vykonávání v praxi zdaleka nedosahují.

2.3 Stažení metrik

2.3.1 Připojení k HTTP API Promethea přes proxy Grafany

HTTP API Promethea v systému pro přepis není z vnější sítě přístupné. Z vnější sítě se ale lze připojit HTTP API Grafany, jehož součástí je proxy, která umožňuje přímý přístup k rozhraní datových zdrojů Grafany (*Grafana documentation*, ©2022, Data source API), mezi něž Prometheus patří. Pro připojení skrz proxy Grafany je vyžadována autentizace pomocí API klíče.

URL rozhraní datového zdroje je určeno podle číselného ID daného datového zdroje v Grafaně. Toto ID je pevně přiděleno a nelze ho v nastavení datového zdroje Grafany změnit na rozdíl od názvu zdroje, jenž může být libovolně nastaven (empiricky ověřeno). Identifikace zdrojů pomocí ID pevně zapsaných v programu tak není vhodná, protože se nelze spoléhat na to, že během vývoje systému ani během života konkrétního nasazení nedojde ke změnám, které by způsobily změnu ID datových zdrojů. HTTP API Grafany ale umožňuje dotazování na ID zdrojů podle jejich názvu (*Grafana documentation*, ©2022, Data source API). Zvoleným řešením tak je, že do kódu aplikace jsou pevně zapsány názvy zdrojů a na jejich ID se aplikace za účelem zjištění URL rozhraní dotazuje.

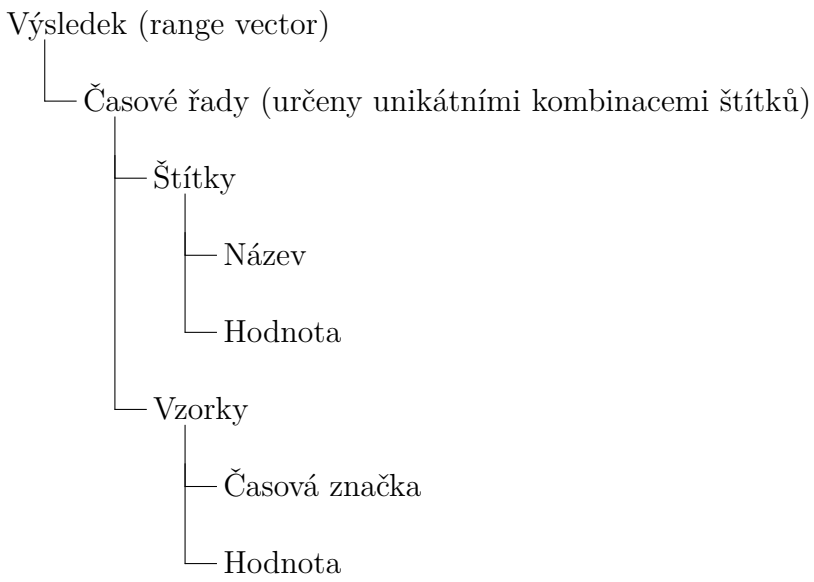
2.3.2 HTTP API Promethea

Z koncových bodů, které HTTP API Promethea poskytuje, byl po vzoru vzorového přehledu použit koncový bod „/api/v1/query_range“ (dále jen „query_range“), který umožňuje získat výsledky PromQL dotazu periodicky vyhodnocovaného přes zadané časové období. Slouží tak k získání informací, které jsou zpracovány z metrik sebraných v zadaném časovém období. Použité parametry HTTP požadavku jsou následující:

- začátek a konec časového období,
- perioda vyhodnocení (krok),
- PromQL dotaz.

(*Prometheus Docs*, © 2014–2022, HTTP API)

Odpověď serveru obsahuje výsledek typu „range vector“ (v API je pro kompatibilitu označován zastaralým názvem „matrix“), jehož struktura je zobrazena na obrázku 2.4. (*Prometheus Docs*, © 2014–2022, HTTP API)



Obrázek 2.4: Struktura výsledku typu range vector
Zdroj: autor na základě (*Prometheus Docs*, © 2014–2022, HTTP API)

Experimentálně bylo ověřeno, že k prvnímu vyhodnocení zadaného dotazu dojde v čase „začátek“. Na dotazy, v odpovědi na něž by se vyskytovala časová řada s více než 11000 vzorky (jedná se o pevný limit rozhraní, který nelze změnit), odpovídá Prometheus chybou („prometheus/prometheus“, 2022, web/api/v1/api.go: řádky 436–441).

Fungování koncového bodu query_range je znázorněno v kapitole 2.3.4.

2.3.3 Klient pro HTTP API Promethea

Pro použití HTTP API Promethea byl hledán klient, který by abstrahoval odesílání HTTP požadavku a korektní zpracování odpovědi pod rozhraním v jazyce Go a vracel odpověď v datových strukturách jazyka Go. Kritérii výběru bylo, aby bylo řešení dostatečně populární (indikace spolehlivosti; měřeno např. podle počtu hvězdiček na stránkách GitHub) a nemělo nadměrné množství závislostí. Vhodného klienta se ale nalézt nepodařilo.

Jedním ze zvažovaných klientů byl oficiální klient v jazyce Go pro HTTP API Promethea („prometheus/client-go“, 2021, api). Tento klient ale nebyl vybrán na základě poznámky, že se jedná o alpha verzi („prometheus/client-go“, 2021).

Jiným zkoumaným klientem byl klient nacházející se v repozitáři nástroje Thanos („thanos-io/thanos“, 2021, pkg/promclient/promclient.go). Ten byl ale odmítnut kvůli množství závislostí. Navíc vyžadoval závislosti, jejichž verzování nebylo kompatibilní s konvencemi jazyka Go a jejichž zahrnutí by způsobovalo potenciální nekompatibilitu s jinými knihovnami, které by je využívaly, například zvažovaným klientem pro HTTP API Lokiho, viz kapitola 2.4.2.

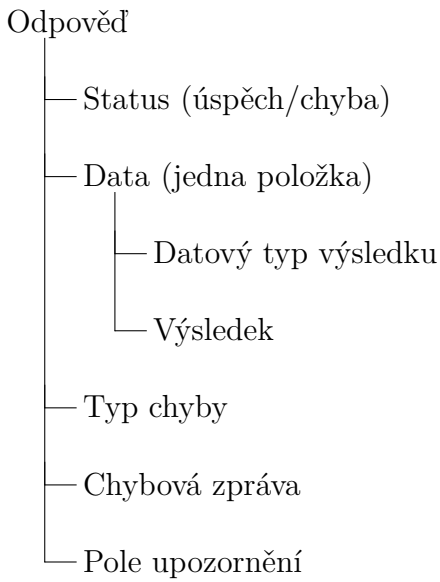
Zvoleným řešením bylo vytvoření vlastního klienta, jehož funkcionalita byla omezena na tu, kterou aplikace využívá. Bylo vyhodnoceno, že implementace vlastního klienta není přílišně náročná a klienta není nutné udržovat vůči změnám API, neboť HTTP API Promethea je verzované a pro verzi 1 je garantována stabilita (*Prometheus Docs*, ©2014–2022, API Stability). Oproti klientu třetí strany je oprava problémů jednodušší: není potřeba vytvářet vlastní odnož (fork) repozitáře nebo v souladu s licencí kopírovat a následně udržovat kód, který je kvůli nadbytečné funkcionalitě rozsáhlejší, než vlastní řešení.

Struktura odpovědi ve formátu JSON koncového bodu query_range, kterou klient parsuje, je zobrazena na obrázku 2.5.

Chyba je vracena jako součást odpovědi ve formátu JSON, klient ale musí počítat i s variantou, že bude vrácena chybová odpověď v jiném formátu (např. prostý text), protože chybou po cestě odpoví někdo jiný než koncový bod API. Aplikace se v případě chyby pokouší parsovat tělo HTTP odpovědi jako JSON a zobrazit chybovou zprávu extrahovanou podle schématu uvedeného na obrázku 2.5.

Pokud tělo HTTP odpovědi není ve formátu JSON, vrací klient jako chybovou zprávu celé tělo odpovědi. Celé tělo odpovědi vrací rovněž v případě, kdy je ve formátu JSON, ale pole „typ chyby“ nebo „chybová zpráva“ jsou prázdná nebo neexistují. Program se pro rozpoznání formátu odpovědi nespoléhá na hlavičku Content-Type v HTTP odpovědi, neboť nemusí být správně vyplněná: např. používaná verze Lokiho nevyplňovala správně Content-Type, když bylo tělo odpovědi ve formátu JSON.

Upozornění obsažená v odpovědi jsou aplikací pro zjednodušení implementace ignorována.



Obrázek 2.5: Struktura odpovědi koncového bodu query_range HTTP API Prometheа

Zdroj: autor na základě (*Prometheus Docs*, © 2014–2022, HTTP API)

2.3.4 Vybraná teorie k dotazování Prometheа

Následující popis vychází z dokumentace (*Prometheus Docs*, © 2014–2022, sekce Querying).

Prometheus rozlišuje dva typy vektorů metrik:

- Instant vektor: Hodnoty z různých časových řad, kterým je dohromady přiřazena jedna časová značka. Vektor má rozměry v prostoru štítků.
- Range vektor: Hodnoty z různých časových řad v určitém časovém intervalu. Každá hodnota má přiřazenou vlastní časovou značku. Vektor má rozměry v prostoru štítků a rozměr v čase.

Dotazy v jazyce PromQL jsou vyhodnocovány k určitému časovému okamžiku, což určuje, z jakého časového období jsou metriky vráceny.

Základním prvkem dotazu PromQL je výběr časových řad, který se provádí specifikací názvu metriky a omezením na vybrané hodnoty štítků:

```
nazev_metriky{key=value, foo!=bar}
```
(2.1)

Pro získání okamžitých hodnot metrik z vybraných časových řad k určitému časovému okamžiku stačí v jazyce PromQL specifikovat výběr těchto řad tak, jak je uvedeno v kódu 2.1. Z každé časové řady spadající do výběru je vybrána poslední metrika nanejvýš 5 minut před čas, ke kterému je dotaz vyhodnocen; v opačném případě je časová řada vynechána. Výsledkem dotazu je instant vektor s časem, ke kterému je dotaz vyhodnocen.

U metrik typu counter (viz 1.4.1) není často zajímavá jejich absolutní hodnota, ale rychlost jejího růstu. Ta se aproximuje z vektoru hodnot metrik následujícím způsobem:

1. Specifikuje se výběr časových řad.
2. Specifikuje se, že je požadován range vektor o určité délce.
3. Na vybraný range vektor je aplikována funkce rate nebo irate aproximující růst hodnoty metriky za sekundu v okamžiku.

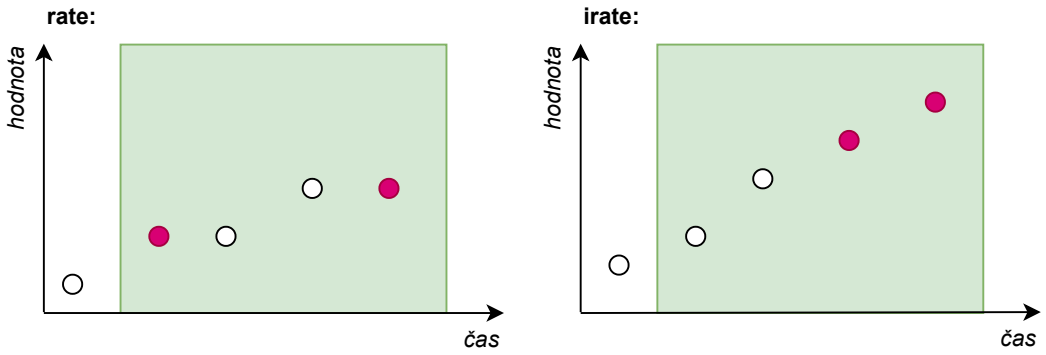
Například (délka vybraného range vektoru je 5 minut):

```
irate(nazev_metriky{key=value, foo!=bar}[5m])
```
(2.2)

Uvedené funkce pro aproximaci míry růstu liší následovně:

- irate: Aproximuje okamžitou míru růstu na základě rozdílu posledních dvou vzorků a jejich vzdálenosti v čase.
- rate: Aproximuje průměrnou míru růstu na základě rozdílu krajních vzorků a délky vektoru. Pokud se krajní vzorek nachází v blízkosti hranice vektoru (přibližně na délku periody sběru dat), je do ní jeho hodnota extrapolována. („prometheus/prometheus“, 2022, promql/functions.go: funkce extrapolatedRate)

Obrázek 2.6 zobrazuje příklad situace, kdy je funkce rate preferována: pokud hodnota metriky roste pomalu a vykazuje změny po delších časových intervalech, než je perioda sběru metrik, a aproximace rychlosti růstu ze sousedních metrik by byla zavádějící.

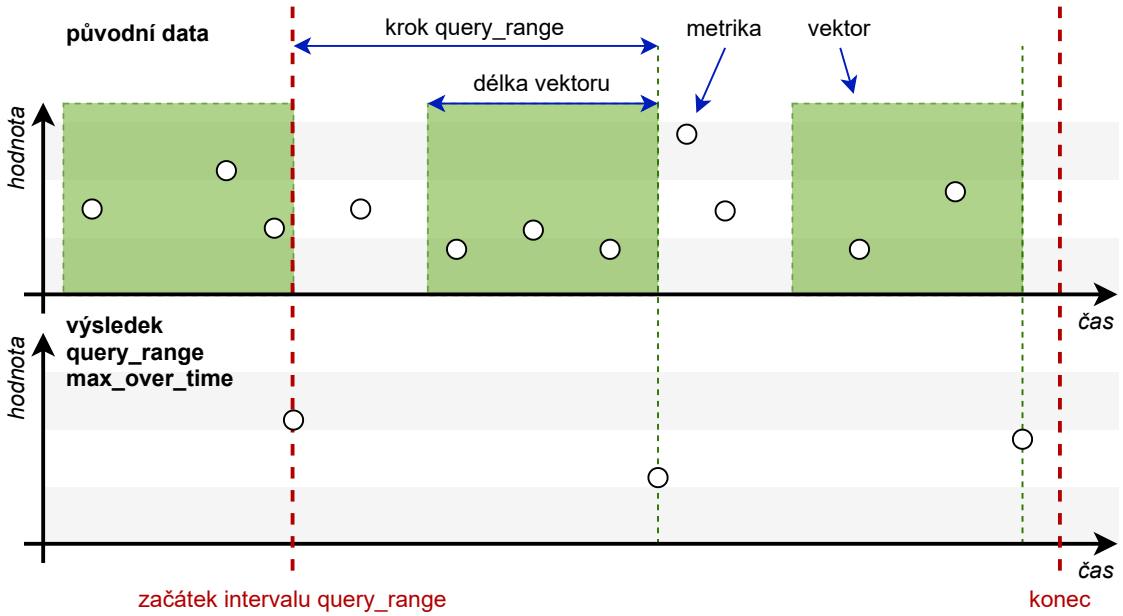


Obrázek 2.6: Srovnání využití funkcí rate a irate
Zdroj: autor na základě (*Prometheus Docs*, ©2014–2022, sekce Querying)

Výsledkem popsaného dotazu pro získání okamžité rychlosti růstu je opět instant vektor s časem, ke kterému byl dotaz vyhodnocen.

Na závěr této kapitoly je ještě znázorněno chování koncového bodu query_range, viz kapitola 2.3.2. Na obrázku 2.7 je zobrazena jedna časová řada v odpovědi na požadavek na koncový bod query_range s dotazem:

```
max_over_time(nazev_metriky[5m])
```
(2.3)



Obrázek 2.7: Příklad použití koncového bodu query_range s dotazem pro výpočet maxima z vektoru metrik v časovém intervalu (znázorněno na jedné časové řadě) Zdroj: autor na základě (*Prometheus Docs*, © 2014–2022, sekce Querying) a vlastního pozorování

V horní části obrázku 2.7 se nachází původní data z časové řady, v dolní výsledná. Funkce max_over_time byla vybrána pro názornost; z každé časové řady v range vektoru vybere nejvyšší hodnotu a jejím výsledkem je instant vektor s časem, ke kterému byl dotaz vyhodnocen. Výsledkem požadavku na koncový bod query_range je pak range vektor obsahující výsledky periodicky vyhodnoceného dotazu.

Požadavkem na koncový bod query_range lze získat data s jinou periodou vzorků, než jakou mají původní sebraná data.

2.3.5 Dotazy ze vzorového přehledu

Grafana umožňuje zobrazit konkrétní dotazy, které slouží ke stahování dat pro vzorový přehled v Grafaně (viz 2.1.1). Z nich byly vybrány dotazy, které se týkají metrik, jež mají být zobrazeny ve výsledných grafech, viz kapitola 2.1.3. Na základě těchto dotazů lze sestavit dotazy, pomocí kterých bude stahovat metriky vyvíjená aplikace. Tato a následující kapitoly se tak věnují rozboru a úpravám těchto dotazů.

Pro zkoumání dotazů a testování variant byl používán nástroj Grafana Explore, viz kapitola 1.6.

Metriky stahované dotazy pochází od dvou poskytovatelů. Prvním je nástroj cAdvisor (Container Advisor) od firmy Google, který zveřejňuje metriky popisující stav kontejnerů a který je integrován v Kubernetes („google/cadvisor“, 2022; *Kubernetes Documentation*, ©2022, Metrics For Kubernetes System Components). Druhým poskytovatelem je nástroj kube-state-metrics, který je součástí projektu Kubernetes a který poskytuje metriky popisující Kubernetes objekty („kubernetes/kube-state-metrics“, 2022). Podle dokumentací obou nástrojů bylo ověřeno, že význam metrik stahovaných dotazy odpovídá předpokladům („google/cadvisor“, 2022, docs/storage/prometheus.md; „kubernetes/kube-state-metrics“, 2022, docs/pod-metrics.md).

Vzhledem k tomu, že dotazů je velké množství, ale řada z nich se liší pouze v detailech, byly pro ukázku v rámci tohoto textu vybrány typové příklady. Aspekty dotazů, které nebyly představeny v kapitole 2.3.4, jsou dále postupně rozebrány. Popis dotazů vychází z dokumentace (*Prometheus Docs*, ©2014–2022, sekce Querying).

Dotaz na okamžité využití paměti

```
sum(
    container_memory_working_set_bytes{
        job="kubelet",
        metrics_path="/metrics/cadvisor",
        cluster="$cluster",
        namespace="$namespace",
        pod="$pod",
        container!="",
        image!=""
    }
) by (container)
```
(2.4)

(„prometheus-community/helm-charts“, 2021, charts/kube-prometheus-stack)

Jako hodnoty štítků se objevují řetězce začínající znakem `$`. Tyto řetězce rozpoznává Grafana jako proměnné, za které dosazuje konkrétní hodnoty dle nastavení přehledu (*Grafana documentation*, ©2022, Variable syntax).

Operátor `sum () by ()` přijme v části „sum“ instant vektor, seskupí časové řady se stejnými hodnotami štítků vyjmenovaných v části „by“ a v každé skupině sečte hodnoty. Výsledkem je instant vektor těchto součtů rozlišených pomocí štítků vyjmenovaných v části „by“; ostatní štítky jsou zahozeny. Část „by“ je volitelná. Nebylo zjištěno, z jakého důvodu je tento operátor v dotazech ze vzorového přehledu použit.

Dotazy na síťové metriky

Jednotlivé dotazy se liší názvem metriky.

```
sum(irate(
    container_network_receive_bytes_total{
        job="kubelet",
        metrics_path="/metrics/cadvisor",
        cluster="$cluster",
        namespace="$namespace",
        pod=~"$pod"
    }[$__rate_interval]
)) by (pod)
```
(2.5)

(„prometheus-community/helm-charts“, 2021, charts/kube-prometheus-stack)

U štítku „pod“ je použit operátor `=~`, který hodnotu porovnává jako regulární výraz. Síťové metriky byly dostupné pouze na úrovni podu, nikoliv kontejneru.

Throttling

```
sum(increase(
    container_cpu_cfs_throttled_periods_total{
        job="kubelet",
        metrics_path="/metrics/cadvisor",
        namespace="$namespace",
        pod="$pod",
        container!="",
        cluster="$cluster"
    }[$__rate_interval]
)) by (container)
/
sum(increase(
    container_cpu_cfs_periods_total{
        job="kubelet",
        metrics_path="/metrics/cadvisor",
        namespace="$namespace",
        pod="$pod",
        container!="",
        cluster="$cluster"
    }[$__rate_interval]
)) by (container)
```
(2.6)

(„prometheus-community/helm-charts“, 2021, charts/kube-prometheus-stack)

V tomto dotazu je použita funkce increase. Její výsledek je dle dokumentace stejný jako výsledek funkce rate vynásobený délkou časového intervalu. Dotaz také obsahuje podíl dvou instant vektorů. Jeho výsledkem je instant vektor, který obsahuje podíly shodných časových řad (časových řad se stejnými štítky).

Okamžitá hodnota nastavení zdrojů přidělených podu v Kubernetes

Jednotlivé dotazy se liší názvem metriky a hodnotou štítku „resource“.

```
sum(
    kube_pod_container_resource_limits{
        job="kube-state-metrics",
        cluster="$cluster",
        namespace="$namespace",
        pod="$pod",
        resource="memory"
    }
)
```
(2.7)

(„prometheus-community/helm-charts“, 2021, charts/kube-prometheus-stack)

Okamžité využití jader procesoru

```
sum(
node_namespace_pod_container:container_cpu_usage_seconds_total:sum_irate{
        namespace="$namespace",
        pod="$pod",
        cluster="$cluster"
    }
) by (container)
```
(2.8)

(„prometheus-community/helm-charts“, 2021, charts/kube-prometheus-stack)

Metrika uvedená v dotazu je tvořena pomocí recording rule (viz 1.4.3) a počítána pomocí následujícího dotazu:

```
sum by (cluster, namespace, pod, container) (
    irate(container_cpu_usage_seconds_total{
        job="kubelet",
        metrics_path="/metrics/cadvisor",
        image!=""
    }[5m])
) * on (cluster, namespace, pod) group_left(node)
topk by (cluster, namespace, pod) (
    1, max by(cluster, namespace, pod, node)
    (kube_pod_info{node!=""})
)
```
(2.9)

(„prometheus-community/helm-charts“, 2021, charts/kube-prometheus-stack/templates/prometheus/rules-1.14/k8s.rules.yaml)

Výraz, kterým je výsledek funkce sum násoben, je prostředek sloužící k doplnění štítku „node“ časovým řadám (Brazil, 2016).

Dotaz pro okamžité využití jader procesoru byl pro konzistenci přepsán do podoby, která přímo používala zdrojovou metriku místo recording rule. Optimalizace

pomocí recording rule nebyla shledána potřebnou. Výraz pro doplnění štítku „node“ z dotazu 2.9 byl vynechán.

2.3.6 Časové řady

Pomocí nástroje Grafana Explore mohlo být zjištěno, jakých hodnot štítky nabývají, a tedy ověřit jejich význam. Pro identifikaci požadovaných časových řad z agenta byly napříč dotazy využity následující štítky:

- pod: Název podu agenta.
- container: Název kontejneru uvnitř podu, v rámci kterého běží agent – „agent“. Tento štítek nebyl využit u metrik dostupných pouze na úrovni podu.

Název podu je unikátní pouze v rámci jmenného prostoru v Kubernetes (viz 1.2.1), pro jednoznačnou identifikaci by tak měl být správně specifikován příslušný jmenný prostor pomocí štítku „namespace“, jako je tomu v uvedených dotazech ze vzorového přehledu. Byl ale vznesen požadavek, aby byl tento štítek ponechán nespecifikovaný, protože se v různých nasazeních systému pro přepis může měnit. Je tak předpokládáno, že název podu agenta je v rámci clusteru unikátní. Výběr ostatních štítků byl ponechán v původní podobě.

Všechny výrazy uvnitř operátoru sum ve všech dotazech vracely pouze jednu časovou řadu. Aby byly snáze odhaleny problémy, byl operátor sum z dotazů odstraněn: pokud neočekávaně přibude časová řada odpovídající dotazům, projeví se ve výstupu a není bez upozornění sečtena. Vyvíjená aplikace ověřuje, že byla vrácena nanejvýš jedna časová řada, v opačném případě se jedná o chybu.

2.3.7 Vzorky metrik

Bylo třeba specifikovat dvě vlastnosti požadavků, které určují, do jaké míry odráží vzorky metrik získané požadavkem původní vzorky obsažené v Prometheu:

- parametr „krok“ koncového bodu query_range,
- způsob zpracování původních vzorků metrik na výsledné v dotazu.

Za účelem určení velikosti parametru „krok“ bylo nejdříve zjišťováno, jaká jeho nejmenší hodnota by byla smysluplná vzhledem k frekvenci, se kterou Prometheus získává aktualizovaná data. Interval, ve kterém zdroj cAdvisor aktualizuje zveřejněné metriky o kontejneru, je ve výchozím stavu dynamický v rozsahu 1 s – 1 min v závislosti na aktivitě kontejneru („google/cadvisor“, 2022, docs/runtime_options.md). U zdroje kube-state-metrics nebyla zjištěna frekvence aktualizace metrik; lze předpokládat, že jsou aktualizovány okamžitě. Prometheus v systému pro přepis je nakonfigurován pro sběr metrik z koncových bodů každých 15 sekund. Z uvedených informací bylo vyhodnoceno, že sběr dat lze považovat za méně frekventovaný než aktualizace na straně zdrojů a za minimální smysluplný krok lze považovat periodu sběru dat o délce 15 sekund.

Jak je uvedeno v kapitolách 1.7 a 2.2, délka trvání vykonávání požadavků se může značně lišit od řádu sekund až po stanovenou hranici 7 dní. Použité rozhraní Promethea omezuje počet vzorků vrácených v časové řadě na 11000, viz kapitola 2.3.2, s parametrem „krok“ o velikosti 15 sekund tak lze získat vzorky pouze z časového období o délce necelých 46 hodin. Je tak otázkou, jestli implementovat dávkové dotazování pro překonání limitu počtu vzorků, nebo se limitu přizpůsobit a v případě rozsáhlých požadavků na přepis zvýšit velikost parametru „krok“.

V případě velkých velikostí parametru „krok“ se nabízí další otázka ohledně informace reprezentované vzorky dat:

- Každý vzorek dat může reprezentovat hodnotu metriky v konkrétním okamžiku, což je případ všech dotazů ze vzorového přehledu. Při velkém parametru „krok“ jsou pak zachyceny izolované vzájemně vzdálené okamžiky a informace mezi nimi je ztracena.

- Každý vzorek dat může reprezentovat hodnoty metrik z rozsáhlejšího časového intervalu pomocí vhodné agregace, např. aritmetického průměru nebo maxima. Informace ze vzorků by tak místo izolovaných okamžiků mohla pokrývat celý časový interval bez navýšení počtu vzorků.

Při konzultaci s vedoucím bakalářské práce bylo specifikováno, že detaily v průběhu požadavku nejsou podstatné a důležitý je především globální průběh. Na základě tohoto tak bylo pro jednoduchost rozhodnuto, že vzorky dat budou reprezentovat izolované okamžiky a stahování dat se přizpůsobí limitu počtu vzorků. Dotazy tak oproti původní podobě nebyly v tomto ohledu upravovány.

U všech dotazů na metriky typu counter bylo z jejich grafů experimentálně ověřeno, že je použití funkce irate vhodné (viz 2.3.4). Délka vektoru, ze kterého je funkce irate počítána, byla stanovena na 5 minut (tatáž délka byla použita v dotazu pro recording rule metriku použitou ve vzorovém přehledu, viz kapitola 2.3.5).

Výsledný krok je počítán dynamicky na základě délky stahovaného časového období a dvou parametrů: minimálního kroku a maximálního počtu vzorků v časové řadě. Minimální krok je jako výsledný krok použit, pokud by počet vzorků nepřesáhl maximum. V opačném případě je použit podíl délky dotazovaného časového intervalu a maximálního počtu vzorků. Výchozí minimální krok byl na základě úvahy uvedené dříve v této kapitole stanoven na 15 sekund. Výchozí maximální počet vzorků v časové řadě byl nastaven na 1000; byla zvolena nižší hodnota, než Prometheus umožňuje, protože vyšší úroveň detailu nebyla požadována. Oba zmíněné parametry jsou nastavitelné uživatelem, viz kapitola 2.6.1.

2.3.8 Hranice stahovaného časového období

Základní hranice časového období, ze kterého jsou metriky staženy, jsou určeny v kapitole 2.2. Bylo rozhodnuto, že bude uživateli umožněno toto časové období volitelně rozšířit, aby mohl v případě potřeby zobrazit metriky ze širšího kontextu období vykonávání požadavku. Výchozí hodnotou tohoto rozšíření bylo zvoleno 5 minut na obě strany intervalu.

Celkový rozsah stahovaného časového období včetně rozšíření je omezen na 7 dní (viz 2.2), aby bylo zabráněno případnému přetížení serveru uživatelem při nastavení nesmyslně velkého rozšíření časového období.

2.4 Stažení logu

K HTTP API Lokiho je přistupováno přes proxy Grafany obdobně jako u Promethea, viz kapitola 2.3.1.

2.4.1 HTTP API Lokiho

Pro stahování logu byl vybrán koncový bod „/loki/api/v1/query_range“, jehož jednou z možností využití je stažení řádek logu ze zadaného časového období. Použité parametry HTTP požadavku jsou pro tento účel následující:

- začátek a konec časového období,
- LogQL dotaz pro filtrování řádek logu (výsledkem dotazu musí být proudy logu),
- směr chronologického řazení,
- limit počtu vrácených řádek.

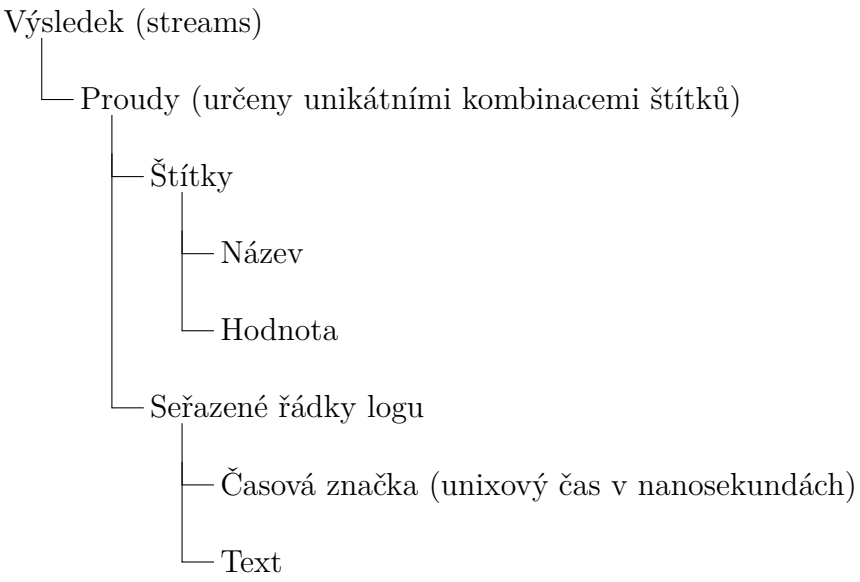
(*Grafana Loki documentation*, ©2022a, HTTP API)

Odpověď serveru obsahuje výsledek typu „streams“, který má strukturu zobrazenou na obrázku 2.8.

Experimentálně byly zjištěny následující vlastnosti API, které se týkají parametrů časového období:

- Pokud $začátek = konec$, vrací požadavek řádky s časovou značkou $začátek = konec$.
- Pokud $začátek < konec$, vrací požadavek řádky s časovou značkou v intervalu $\langle začátek, konec)$.

Tvorba dotazu pro vyfiltrování požadovaného logu a výsledné proudy jsou popsány v kapitole 2.4.3. Výběrem časového období se zabývá kapitola 2.4.4. Chronologické řazení bylo vybráno vzestupné, aby mohlo být použito ve výstupním souboru. Limit počtu vrácených řádek brání stažení všech řádek v požadovaném časovém období jedním dotazem, řešením se zabývá kapitola 2.4.5.



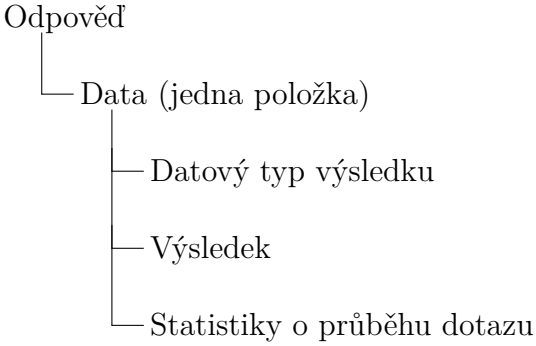
Obrázek 2.8: Struktura výsledku typu streams
Zdroj: autor na základě (*Grafana Loki documentation*, ©2022a, HTTP API)

2.4.2 Klient pro HTTP API Lokiho

Obdobně jako u Promethea byl neúspěšně hledán klient, který by zjednodušil dotazování HTTP API Lokiho z jazyka Go, viz 2.3.3. Zvažovaným klientem byl klient používaný oficiálním nástrojem Lokiho s názvem LogCLI („grafana/loki“, 2022, pkg/logcli/client). Nebyl ale použit kvůli množství závislostí a problematickým závislostem (viz 2.3.3).

Stejně jako u Promethea (viz 2.3.3) byl i v tomto případě implementován vlastní klient. HTTP API Lokiho je stejně jako HTTP API Promethea verzované (*Grafana Loki documentation*, ©2022a, HTTP API), takže lze předpokládat, že nedochází ke zpětně nekompatibilním změnám.

Struktura odpovědi ve formátu JSON koncového bodu query_range, kterou klient parsuje, je zobrazena na obrázku 2.9.



Obrázek 2.9: Struktura odpovědi koncového bodu query_range HTTP API Lokiho
Zdroj: autor na základě (*Grafana Loki documentation*, ©2022a, HTTP API)

Odpovědi HTTP API Lokiho jsou kompatibilní s HTTP API Promethea (viz 1.5.3, 2.3.3), takže je pro práci s nimi v programu implementována jednotná sada funkcí a datových struktur. Odpověď Lokiho na rozdíl od Promethea neobsahuje upozornění a v případě chyby odpovídá chybovou hláškou ve formě prostého textu (experimentálně ověřeno). Statistiky o průběhu dotazu byly ignorovány.

2.4.3 Proudy logu

Stejně jako u metrik byl pro ladění dotazu pro získání požadovaných proudů logu využit nástroj Explore v Grafaně (viz 1.6). Pomocí něj bylo možné vypsat seznam všech štítků, jimiž jsou řádky logu v Lokim v systému pro přepis označeny. Také bylo možné vyzkoušet dotaz na log s určitými štítky a zjistit, kolik proudů logu se ve výstupu nachází a jakými dalšími štítky jsou opatřeny.

Těmito metodami byly zjištěny štítky, pomocí kterých bylo možné získat proudy logu z agenta, který vykonával požadavek:

- namespace: Jmenný prostor v Kubernetes.
- pod: Název podu.
- container: Název kontejneru v podu.

Stejně jako u metrik byl štítek „namespace“ ve výsledném dotazu nevyužit, viz kapitola 2.3.6. Kontejner v podu, který vykonává činnost agenta, se nazývá „agent“.

Výsledný dotaz tak je (ostré závorky označují parametr):

```
{pod="<název podu>", container="agent"}
```
(2.10)

Z dalších štítků byl shledán podstatným pouze štítek „stream“, který rozlišuje log ze standardního výstupu a standardního chybového výstupu, což může být užitečná informace pro uživatele za účelem vyhledávání v logu.

Kód zpracovávající získaný log pro jistotu předpokládá libovolné množství výstupních proudů, na základě experimentů se ale ve výstupu dotazu nachází maximálně dva proudy rozlišené podle hodnoty štítku „stream“; zbylé štítky měly zřejmě charakter doplňkových informací k existujícím proudům.

2.4.4 Hranice stahovaného časového období

I zde, stejně jako u metrik (viz 2.3.8), je uživateli umožněno volitelně rozšířit základní hranice časového období, ze kterého je log stahován. Tento parametr je nezávislý na parametru velikosti rozšíření časového období, ze kterého jsou stahovány metriky. Pro výchozí nastavení byla na rozdíl od metrik zvolena nejnižší možná velikost rozšíření, protože by řádky logu ze širšího kontextu mohly být ve výsledném souboru příliš rušivé, pokud by je uživatel nepotřeboval.

Byly identifikovány následující důvody, proč není vhodné, aby velikost rozšíření časového období byla nulová (pro kontext: log je sbírán průběžně a časové značky logu nastavuje Promtail na stroji s agentem podle času sběru, viz kapitola 1.5.2):

- Je vhodnější počítat s malým zpožděním sběru logu, protože i velmi malé zpoždění by mohlo ovlivnit obsah výstupu.
- Časy naplánování požadavku a ukončení požadavku, které slouží jako základní hranice časového období stahovaných dat (viz 2.2), jsou nastavovány plánovačem, zatímco časové značky logu nastavuje Promtail na stroji s agentem. Protože se agent a plánovač mohou nacházet na odlišných strojích, může mezi nimi docházet k časovému rozchodu, pro který je vhodné udělit prostor. Je ale předpokládáno, že hodiny strojů v clusteru jsou synchronizovány, takže by rozchod měl být minimální.
- Lze očekávat, že agent bude produkovat relevantní log i krátce po ukončení přepisu.

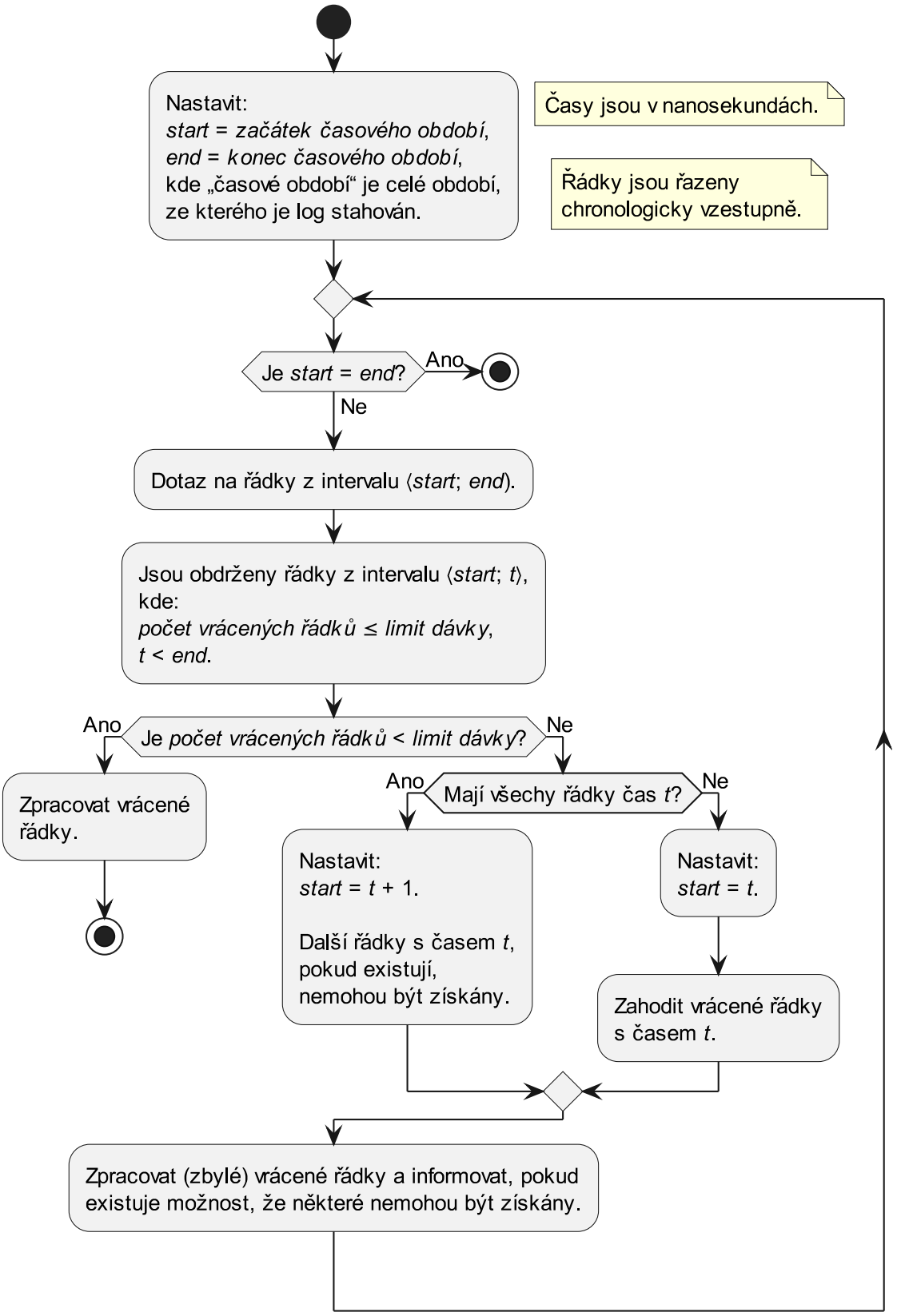
Zvoleno bylo výchozí rozšíření hranic o velikosti 2 sekund na obě strany, které se při testování jevilo dostatečným; na základě budoucích zkušeností může být upraveno. Stejně jako u metrik (viz 2.3.8) je celkový rozsah stahovaného časového období včetně rozšíření omezen na 7 dní.

2.4.5 Dávkové dotazování

Vzhledem k tomu, že množství logu může být libovolně veliké, je vhodné se dotazovat na jeho jednotlivé části (dávky) a nikoliv na celý jeho objem, aby mohly být řádky logu přijaty a zpracovány postupně a nedošlo k zahlcení paměti zařízení, na kterém aplikace pro stahování dat běží. Dávkové dotazování je zároveň vynuceno tím, že počet řádek vracených API na jeden požadavek je povinně limitován jeho parametrem „limit“. Maximální hodnota parametru „limit“, kterou Loki na straně serveru ve výchozím nastavení dovoluje, je 5000 řádek (*Grafana Loki documentation*, © 2022a, Configuring Grafana Loki); toto nastavení je použito v systému pro přepis.

Navržený a posléze použitý algoritmus pro dávkové dotazování je popsán na obrázku 2.10. Zpracování řádek znamená v tomto případě jejich zapsání do výstupního souboru. Zahazování řádek je prováděno, neboť pokud odpověď podlehne limitu v čase t, nemusely být vráceny všechny požadované řádky s časem t.

Důležitý předpoklad uvedeného algoritmu je, že v žádné odpovědi API *nechybí napříč všemi proudy odpovídajícím LogQL dotazu žádný řádek z intervalu* $\langle start, t)$. Platnost tohoto předpokladu ale dokumentace API Lokiho explicitně nezmiňuje. Při rešerši bylo zjištěno, že oficiální nástroj LogCLI taktéž umožňuje dávkové stahování logu (*Grafana Loki documentation*, © 2022b, LogCLI). Nástroj má otevřený zdrojový kód a při jeho analýze bylo zjištěno, že jeho implementace dávkového stahování funguje obdobným způsobem a závisí tak na stejném předpokladu („grafana/loki“, 2022, pkg/logcli/query/query.go: řádky 93–151). Kód nástroje LogCLI nebyl přímo využit, protože logika dávkového dotazování je v něm úzce svázaná s konkrétním



Obrázek 2.10: Dávkové dotazování na log
Zdroj: autor

využitím a musel by být do značné míry modifikován (licenční záležitosti nebyly zvažovány).

U dotazování může teoreticky dojít k situaci, kdy všechny vrácené řádky v dávce mají stejnou časovou značku. API v takovém případě neumožňuje dotázání na zbylé řádky s danou časovou značkou (pokud existují – aplikace toto ale pro jednoduchost dále nezjišťuje). Aplikace je tak místo toho přeskočí, zapíše upozornění do výstupního souboru a pokračuje od dalšího okamžiku. LogCLI ve stejné situaci končí chybou („grafana/loki“, 2022, pkg/logcli/query/query.go: řádky 125–131). V praxi by k této situaci nemělo docházet, neboť by to znamenalo, že bylo vygenerováno velmi velké množství řádek logu v jeden časový okamžik.

2.5 Výstup aplikace

Na základě návrhu výstupu aplikace v kapitole 2.1.3 byl zvolen výstup v podobě archivu ve formátu ZIP, který obsahuje následující soubory:

- log.txt: Log a doplňující informace.
- metrics.html: HTML dokument s interaktivními grafy metrik.
- request.json: Objekt požadavku převedený do formátu JSON.

Výběr formátu archivu je vysvětlen a obsah souborů popsán v následujících podkapitolách.

2.5.1 Výběr formátu archivu

Formát komprimovaného archivu byl vybírán mezi formátem ZIP a formátem tar s kompresí gzip, které patří mezi běžné a jsou podporovány standardní knihovnou jazyka Go. Jedním z rozdílů formátů ZIP a tar je, že zápis do archivu tar vyžaduje, aby byla před zápisem každého souboru zapsána hlavička specifikující jeho velikost („archive/tar“, 2022), zatímco zápis souboru do archivu ZIP předchozí znalost velikosti souboru nevyžaduje („archive/zip“, 2022). Formát ZIP je tak z tohoto pohledu vhodnější pro ukládání potenciálně rozsáhlého logu neznámé velikosti, protože ho lze po částech ukládat přímo do vytvářeného archivu na disku. V jiných ohledech nebyl mezi formáty zaznamenán pro účely aplikace významný rozdíl.

2.5.2 Log

Log je uložen do textového souboru. Na obrázku 2.11 je zobrazena ukázka (některé údaje byly anonymizovány). Řádky logu jsou seřazeny vzestupně v chronologickém pořadí. Každá řádka je opatřena časovou značkou, která je relativní vůči času zahájení přepisu, pokud již přepis započal (před časem zahájení přepisu je negativní), v opačném případě je relativní vůči času naplánování vykonání požadavku. Dále jsou řádky logu opatřeny označením proudu: „O“ pro standardní výstup a „E“ pro

standardní chybový výstup. Mezi řádky logu jsou vloženy milníky označující časy naplánování vykonání požadavku, zahájení a ukončení přepisu a ukončení požadavku. Pokud při stahování logu nemohly být získány všechny řádky (kvůli limitu dávky a stejným časovým značkám, viz kapitola 2.4.5), je o tomto mezi řádky logu vloženo varování.

2.5.3 Metriky

Pro zobrazení grafů metrik byla z důvodu přehlednosti, přenositelnosti a podpory interaktivity zvolena forma HTML dokumentu.

Pro generování dokumentu aplikací byla zprvu použita knihovna go-echarts („go-echarts/go-echarts“, 2022), která umožňuje v jazyce Go definovat dokument s interaktivními grafy a uložit ho. Grafy jsou touto knihovnou interně vytvářeny pomocí knihovny jazyka JavaScript s názvem Apache ECharts (*Apache ECharts*, © 2017–2022). Možnosti knihovny ECharts byly vyhovující, použití knihovny go-echarts ale bylo oproti přímému použití ECharts v Javascriptu shledáno uživatelsky nepohodlným a omezujícím. Na základě toho byla knihovna go-echarts nahrazena vlastním řešením pro generování dokumentu: součástí programu je HTML šablona, která obsahuje kód v JavaScriptu pro tvorbu grafů, do níž jsou data vkládána s využitím šablonovacího balíku jazyka Go („html/template“, 2022). Knihovna ECharts je vložena přímo do HTML dokumentu, aby byl dokument kompaktním celkem bez externích závislostí.

Grafy ve výstupu odpovídají grafům navrženým v kapitole 2.1.3. Ukázku zobrazuje obrázek 2.12. Svislými čarami jsou v grafech obdobně jako u logu vyznačeny milníky v průběhu požadavku. Časy jsou převedeny do lokální časové zóny zařízení, na kterém byl program pro stahování dat spuštěn. Grafy umožňují přibližování[1], posouvání a ukládání ve formátu PNG.

Způsob značení milníků v průběhu požadavku není, jak je zřejmé z obrázku 2.12, za všech okolností ideální. Díky možnosti přibližování je ale možné zobrazit konkrétní popisek nezastíněný ostatními, proto bylo od hledání řešení v rámci této práce upuštěno.

2.5.4 Objekt požadavku

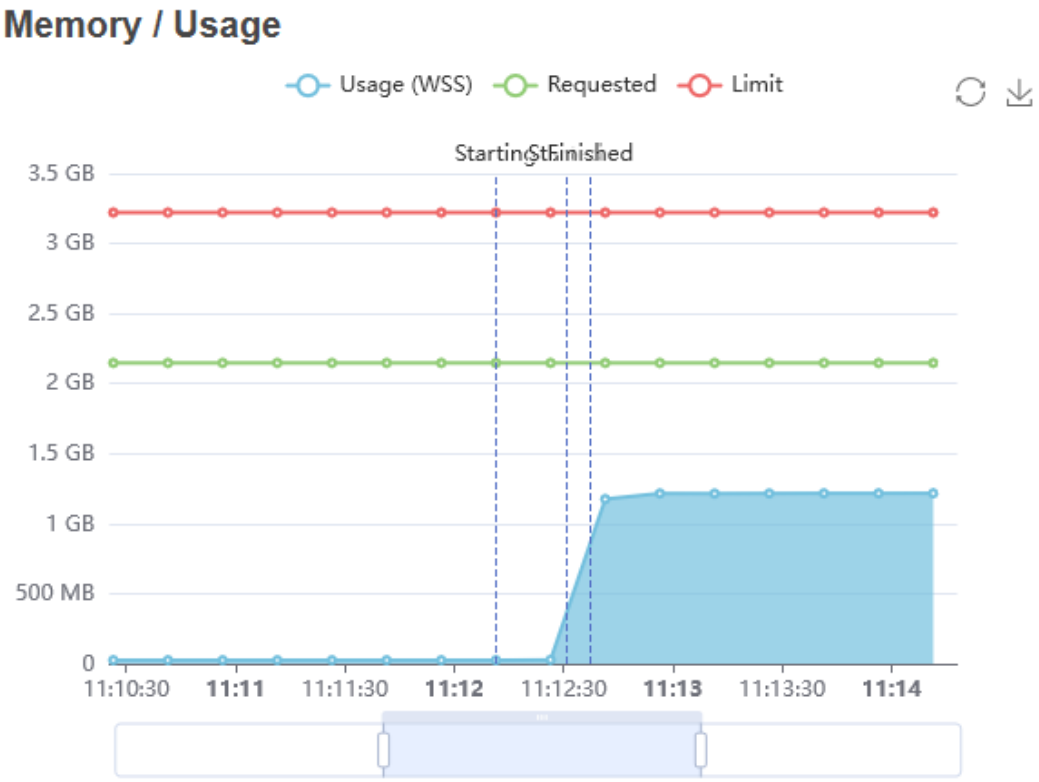
Jako řešení uložení informací obsažených v objektu požadavku bylo zvoleno uložení objektu požadavku ve formátu JSON. Tento formát je čitelný pro člověka a implementace byla snadná, neboť převod objektu do formátu JSON byl v systému pro přepis již implementován.

Nedostatkem řešení je, že objekt požadavku obsahuje časové značky zapsané v unixovém čase, které ve výsledném souboru zůstávají v původní podobě. Vzhledem k tomu, že podstatné časové údaje jsou dostupné ve zformátované, pro člověka čitelné podobě v souboru s grafy metrik, bylo rozhodnuto tento nedostatek prozatím neřešit.

[1] V rámci práce je sice zmiňováno, že není požadována vysoká úroveň detailu, knihovna pro zobrazení grafů ale byla vybrána s ohledem na flexibilitu a podpora přibližování nezpůsobila nadbytečnou zátěž na vývoj.

```
-0:12 E| [2022-03-08T10:12:17.8755274Z] [Information] [[redacted].command.service.run] Starti
-0:12 E| [2022-03-08T10:12:17.8797254Z] [Information] [[redacted].command.service.run] Starti
-0:12 E| [2022-03-08T10:12:18.0192719Z] [Information] [[redacted].api.service] Listening on a
-0:11 E| [2022-03-08T10:12:19.0984601Z] [Debug] [[redacted]api.service] Ping from ipv4:127.0
-0:11 E| [2022-03-08T10:12:19.1020167Z] [Debug] [[redacted].api.service] Ping from ipv4:127.0
-0:11 O| INFO : 2022/03/08 10:12:19.112261 request 5a0e857d-5220-4079-81cb-cc122f8f1df4:
-0:10 E| [2022-03-08T10:12:20.0514253Z] [Information] [[redacted].api.core.app] Starting app
-0:02 E| [2022-03-08T10:12:28.4961035Z] [Debug] [[redacted]api.core.app] Opening app { "star
-0:00 E| [2022-03-08T10:12:29.5723237Z] [Debug] [[redacted].api.core.app] Running app
-0:00 E| [2022-03-08T10:12:29.6864123Z] [Information] [[redacted].api.pipe] { "init": { "type
 0:00 --- REQUEST STARTED: 2022-03-08T11:12:30.534+01:00 ---
 0:06 E| [2022-03-08T10:12:37.3606134Z] [Debug] [[redacted].api.core.app] Completed app
 0:06 --- REQUEST FINISHED: 2022-03-08T11:12:37.365+01:00 ---
 0:06 O| INFO : 2022/03/08 10:12:37.365823 request 5a0e857d-5220-4079-81cb-cc122f8f1df4
```

Obrázek 2.11: Ukázka výstupního souboru s logem
Zdroj: autor



Obrázek 2.12: Ukázka grafu ze souboru s grafy metrik
Zdroj: autor

2.6 Uživatelské rozhraní

2.6.1 Ovládání aplikace

Aplikace je na základě návrhu v kapitole 2.1.4 ovládána pomocí rozhraní v příkazové řádce. K ovládání slouží jediný příkaz, který má tvar:

```
debug-request <ID požadavku> [volby…]
```
(2.11)

Volby aplikace (flags) jsou popsány dále.

Připojení a autentizace

- URL koncového bodu rozhraní systému pro přepis.
- Přihlašovací údaje do systému pro přepis.
- API klíč do Grafany v systému pro přepis. Není vyžadován, pokud je stahován pouze objekt požadavku.
- Cesta ke konfiguračnímu souboru s údaji pro připojení.

Volby pro připojení a autentizaci lze nastavit jako parametr příkazu, přes proměnné prostředí nebo přes konfigurační soubor ve formátu YAML. Tyto dodatečné možnosti zjednodušují použití aplikace a byly implementovány tak, aby byly kompatibilní s ostatními existujícími nástroji pro práci se systémem pro přepis. Volby jsou aplikovány a přepisovány v následujícím pořadí:

1. proměnné prostředí,
2. konfigurační soubory,
3. volby nastavené v příkazu.

Výstup

- Cesta pro uložení výstupního souboru, nebo cesta ke složce, do které má být výstupní soubor uložen pod výchozím názvem. Výchozí název je ve formátu `debug_request_<ID požadavku>.zip` (ostré závorky označují parametr). Pokud soubor s názvem výstupního souboru již existuje, je přepsán.
- Maximální dovolená velikost výstupního souboru nebo nastavení neomezené. Výchozí hodnotou byl zvolen 1 GB; na základě budoucích zkušeností může být upravena. Účelem této volby je ochrana uživatele před nechtěným zaplněním disku.

Metriky

- Vynechání stahování a ukládání metrik.
- Velikost rozšíření hranic časového období, ze kterého jsou stahovány metriky, před čas naplánování vykonání požadavku a za čas ukončení požadavku. Výchozí jsou obě 5 min, viz kapitola 2.3.8.
- Minimální vzorkovací perioda metrik. Výchozí je 15 s, viz kapitola 2.3.7.
- Maximální počet vzorků v časové řadě nebo nastavení neomezeného. Výchozí maximální počet vzorků je 1000, viz kapitola 2.3.7.

Log

- Vynechání stahování a ukládání logu.
- Velikost rozšíření hranic časového období, ze kterého je stahován log, před čas naplánování vykonání požadavku a za čas ukončení požadavku. Ve výchozím nastavení jsou obě 2 s, viz kapitola 2.4.4.
- Limit počtu řádků získaných jedním dotazem pro Lokiho, viz kapitola 2.4.5. Výchozí je 5000, tedy maximální počet, který Loki na straně serveru ve výchozím nastavení dovoluje.

Příkaz poskytuje také volbu pro výpis nápovědy.

Maximální délka časového období stahovaných dat o velikosti 7 dní není nastavitelná, neboť jedním z důvodů její přítomnosti je ochrana serveru před přetížením.

Detaily způsobu zadávání voleb jsou popsány v nápovědě příkazu; v tomto textu byly pro stručnost vynechány.

2.6.2 Implementace rozhraní

Rozhraní aplikace je implementováno pomocí knihovny Cobra („github.com/spf13/cobra“, 2022), která umožňuje mimo jiné vytváření hierarchických struktur příkazů[2], parsování argumentů a voleb (flags) zadaných uživatelem a generování nápovědy. Nastavení parametrů aplikace pomocí konfiguračních souborů a proměnných prostředí je implementováno pomocí knihovny Viper („github.com/spf13/viper“, 2021), která je s knihovnou Cobra provázána. Obě knihovny byly vybrány, protože jsou využívány i dalšími programy pro práci se systémem pro přepis.

[2]Toho bylo využito na počátku vývoje, kdy byla aplikace součástí jiného programu pro práci se systémem pro přepis. Výsledná aplikace je ale nakonec samostatná a tvořena pouze jedním příkazem.

2.7 Doplňující poznámky a specifikace

2.7.1 Náročnost dotazů na systém

Při ladění v Grafana Explore nebylo zaznamenáno, že by vykonávání dotazů bylo časově náročné (ze subjektivního pohledu proběhly prakticky okamžitě). Dotazy vytvářené aplikací způsobují pouze jednorázovou zátěž, navíc není přepokládáno, že by byla frekventovaná. Protože se náročnost dotazů nejevila problematickou, nebyla v rámci práce dále zkoumána. Pro ochranu proti potenciálně náročným dotazům na nadměrně rozsáhlá časová období byla nicméně implementována ochrana, viz kapitoly 2.3.8 a 2.4.4.

2.7.2 Časové zóny

Pokud není v textu uvedeno jinak, byly časové značky na vstupech a výstupech rozhraní vyjádřeny ve formátech, které jsou jednoznačně vztaženy buď k UTC, nebo k jiné explicitně vyjádřené časové zóně (používány byly norma RFC3999 a unixový čas). K záměně časových zón tak v těchto případech nemohlo dojít. S časem bylo pracováno pomocí balíku time ve standardní knihovně jazyka Go, který pracuje s časovými značkami dohromady s časovými zónami („time“, 2022). Na výstupu aplikace jsou časové značky vyjádřeny v lokální časové zóně.

2.7.3 Chování aplikace při chybě

Bylo rozhodnuto, že pokud při běhu programu dojde k chybě (typicky k chybě připojení k serverům, ze kterých jsou čtena data), je běh programu ukončen. Pro jednoduchost je v takovém případě nekompletní výstupní soubor smazán a uživateli je zobrazena chybová hláška popisující konkrétní příčinu.

Protože má ze stahovaných dat potenciálně největší objem log, zatímco objem ostatních dat je omezený (objekt požadavku má malou velikost, a objem metrik je omezen maximální počtem vzorků v časové řadě, viz kapitola 2.3.7), je log stahován jako poslední, aby při chybě během stahování ostatních dat nedošlo ke zmaření času stráveného stahováním logu.

2.8 Testování

Funkčnost vyvíjené aplikace proti systému pro přepis byla testována zejména proti existujícímu nasazení na testovacím clusteru, který se nacházel na TUL. Tento cluster byl sdílen s dalšími vývojáři. Později byla pro účely testování nasazena privátní instance systému pro přepis, viz kapitola 3.

Pro zadávání požadavků na přepis byla k dispozici utilita, které se pro spuštění přepisu předaly jako vstup údaje pro připojení k API jádra systému pro přepis a zvukový soubor určený k přepisu. Ve webovém rozhraní jádra pak bylo možné

zjistit ID zadaného požadavku a použít ho jako vstup vyvíjené aplikace pro stahování diagnostických dat.

Aplikace byla testována formou smoke testu – spuštěním proti dříve vytvořenému požadavku. Výstup aplikace byl ověřen proti informacím zobrazeným v Grafaně a ve webovém přehledu jádra systému pro přepis. Testován byl pouze běh proti úspěšně dokončenému požadavku. Podrobné integrační testy nebyly vedoucím bakalářské práce vyžadovány.

Při testování dokončené aplikace proti nasazení na testovacím clusteru nebyly nalezeny problémy. Při pozdějším testování proti vlastnímu nasazení (verze systému pro přepis byla jiná, než při předchozím testování vůči testovacímu clusteru) se vyskytl problém, který je popsaný v kapitole 3.3.

3 Nasazení systému pro přepis pro otestování vyvinuté aplikace

Posledním cílem bakalářské práce bylo nasadit systém pro přepis a otestovat proti němu vyvinutou aplikaci pro stahování diagnostických dat. Oproti původnímu zadání práce nakonec nebyl pro tento účel poskytnut cluster, ale k dispozici byl pouze jeden stroj. Postup nasazení systému pro přepis byl ale v principu podobný postupu nasazení na více strojů.

Přidělený stroj byl umístěn na půdě TUL a byl ovládán vzdáleně. Nacházel se pod správou nástroje MAAS (Metal as a Service), což je open-source nástroj vyvíjený společností Canonical pro správu hardware za účelem tvorby datového centra (*MAAS Documentation*, ©2022, About MAAS). Na přidělený stroj bylo nejprve potřeba prostřednictvím MAAS nainstalovat operační systém. Poté bylo možné se ke stroji připojit pomocí SSH. Nasazení systému pro přepis bylo provedeno pomocí připraveného automatizovaného řešení vyvíjeného autory systému pro přepis, které využívá nástroj Ansible. K němu bylo ale potřeba doplnit konfiguraci a odstranit chyby, neboť se řešení nacházelo ve stádiu vývoje a nebylo zcela odladěno.

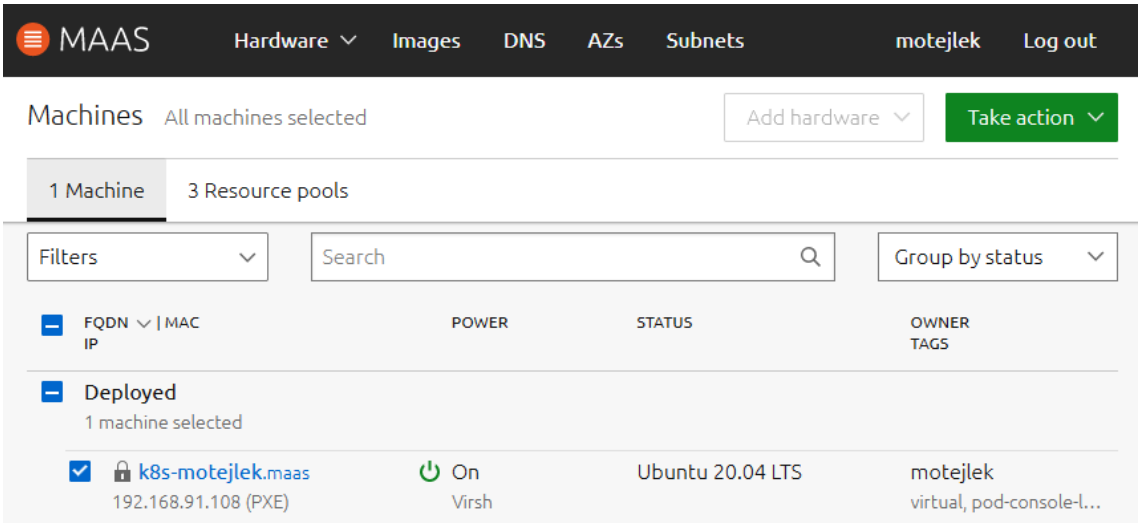
V následujících podkapitolách jsou přiblíženy zmíněné kroky a na závěr je popsán výsledek nasazení a testování.

3.1 Příprava stroje

Pro správu poskytnutého stroje bylo používáno webové rozhraní nástroje MAAS (na obrázku 3.1), které běželo na řídícím stroji (controller) MAAS clusteru, jehož byl přidělený stroj součástí. Pro přístup k nástroji byl poskytnut uživatelský účet. Rozhraní zobrazuje mimo jiné seznam dostupných strojů a jejich stav a umožňuje se stroji provádět akce.

Postupy a významy stavů stroje uvedené dále vychází z dokumentace (*MAAS Documentation*, ©2022, MAAS concepts and terms reference, How to deploy machines).

Stroj se po převzetí nacházel ve stavu „Ready“, který označuje nepoužívaný stroj připravený k nasazení operačního systému. Pro instalaci operačního systému slouží akce „Deploy“ (nasazení), která nainstalovaný systém zároveň nakonfiguruje včetně přístupu přes SSH. K tomu, aby mohlo být nasazení provedeno, bylo nejprve nutné do účtu v MAAS importovat veřejný SSH klíč, který pak byl používán k přístupu ke stroji (*MAAS Documentation*, ©2022, How to manage user accounts). Po zvolení



Obrázek 3.1: Ukázka webového rozhraní MAAS
Zdroj: autor

akce „Deploy“ byl umožněn výběr linuxové distribuce a jádra. Jedinou dostupnou distribucí bylo „Ubuntu 20.04 LTS“; na výběru jádra nezáleželo, takže bylo zvoleno výchozí dostupné. Po potvrzení parametrů nasazení začala instalace a konfigurace operačního systému a stroj byl přesunut do stavu „Deploying“. Po dokončení nasazení stroje se jako jeho stav ve webovém rozhraní zobrazil název nainstalovaného operačního systému a stroj byl připraven k použití.

Volbou „Release“, která stroj uvede zpět do stavu „Ready“, a následným novým nasazením operačního systému bylo možné vrátit stroj do výchozího stavu. Toho bylo využíváno při neúspěšných nasazeních systému pro přepis, která by bylo obtížné opravit, a pro konečné otestování odladěného řešení pro automatizaci nasazení.

Přes síť byl stroj dostupný pod doménou k8s_motejlek.maas, jež mu byla v systému MAAS nastavena. (*MAAS Documentation*, ©2022, How to manage machines)

3.2 Nasazení systému pro přepis

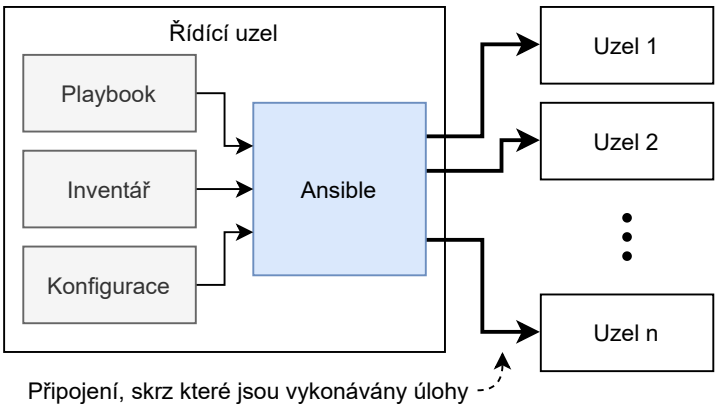
3.2.1 Popis automatizace nasazení pomocí nástroje Ansible

Jak bylo zmíněno v úvodu kapitoly 3, autory systému pro přepis je vyvíjeno řešení pro automatizaci nasazení systému pro přepis využívající nástroj Ansible. Prováděné nasazení je přiblíženo v kapitole 1.8.

Ansible je open-source nástroj pro automatizaci úloh zaštiťovaný společností Red Hat, který umožňuje definovat a spouštět komplexní úlohy na skupině vzdálených strojů. Jeho fungování znázorňuje obrázek 3.2.

Pro vykonání úloh jsou jako vstup nástroji předány:

- Playbook: Zahrnuje definice vykonávaných úloh a používané soubory (např. konfigurační soubory nasazovaných nástrojů). Je tvořen soustavou souborů,



Obrázek 3.2: Schéma fungování nástroje Ansible
Zdroj: autor

přičemž soubory specifikující úlohy jsou zapsány pomocí syntaxe Ansible do formátu YAML. (*Ansible Documentation*, 2022, Intro to playbooks)

- Inventář (inventory): Obsahuje údaje pro připojení ke strojům, hodnoty proměnných přiřazených strojům a popisuje zařazení strojů do skupin. Jednou z možností zápisu je soubor ve formátu YAML. (*Ansible Documentation*, 2022, How to build your inventory)

- Soubor s konfigurací nástroje Ansible ovlivňující jeho chování. (*Ansible Documentation*, 2022, Ansible Configuration Settings)

Součástí vyvíjeného řešení pro automatizaci nasazení clusteru systému pro přepis pomocí Ansible jsou připravený playbook a konfigurace Ansible. Inventář je třeba vytvořit vlastní pro konkrétní případ nasazení.

Ansible je nainstalován pouze na uzlu, ze kterého jsou úlohy spouštěny. Na straně uzlů, na kterých jsou úlohy vykonávány, nevyžaduje žádného agenta; Ansible se ke každému uzlu připojuje způsobem definovaným v inventáři (typicky přes SSH) a úlohy vykonává přes toto spojení. („How Ansible Works“, © 2020)

Základním stavebním prvkem playbooku je tzv. úloha (task), v jejíž definici je specifikován tzv. modul – malý program vykonávající požadovanou úlohu – a jeho argumenty. Ansible obsahuje řadu zabudovaných modulů; další moduly lze přidat v rámci doinstalovatelných kolekcí (collection). Nasazení systému pro přepis vyžaduje kolekci kubernetes.core, která umožňuje pracovat s Kubernetes API (*Ansible Documentation*, 2022, kubernetes.core.k8s module). (*Ansible Documentation*, 2022, Intro to playbooks, Using collections, Ansible.Builtin)

V rámci playbooku lze nastavit, na kterých strojích jsou provedeny které úlohy a jakým způsobem je jejich vykonávání koordinováno. Podporováno je např. také podmíněné vykonávání úloh a vykonávání smyček. Lze specifikovat chování, pokud vykonávání úlohy skončí chybou. (*Ansible Documentation*, 2022, Intro to playbooks, Conditionals, Loops, Error handling in playbooks)

Ansible umožňuje parametrizaci úloh pomocí proměnných, které mohou být nastaveny v playbooku, v inventáři nebo v průběhu vykonávání úloh. Každý stroj má oddělenou sadu hodnot proměnných, která je používána při vykonávání úloh na daném stroji. Proměnné mohou také sloužit ke generování souborů pomocí šablonovacího systému Jinja2. To je u systému pro přepis používáno k doplňování údajů do konfiguračních souborů ve formátu YAML, které jsou používány řadou nástrojů. (*Ansible Documentation*, 2022, Using Variables, Templating (Jinja2))

Konfigurace nasazení systému pro přepis není vzhledem k jejímu rozsahu a specifičnosti v tomto textu rozebírána příliš podrobně. Uveden je pouze hrubý přehled vybraných parametrů:

- Údaje pro připojení k uzlům pro Ansible (plugin pro připojení, uživatelské jméno atd.). (*Ansible Documentation*, 2022, Special Variables)
- Konfigurace clusteru:
  - Přepínač „vysoké dostupnosti“ ovlivňující mimo jiné počty replik komponent a tím tak celkové výpočetní nároky.
  - Adresa repozitáře a přístupové údaje (obsahuje obrazy komponent jádra a služby pro přepis).
- Konfigurace komponent:
  - Zapnutí/vypnutí.
  - Verze.
  - Parametry komponent (hesla, velikosti úložišť, …).
- Konfigurace uzlů:
  - Přiřazení do skupin: master, běžné a administrativní, viz kapitola 1.8. Skupiny se vzájemně nevylučují.
  - Štítky a označení „taint“ v Kubernetes.

Parametry clusteru a komponent jsou specifikovány ve formě proměnných přiřazených skupině „all“, což je speciální skupina, jíž jsou všechny stroje součástí. Parametry uzlů mohou být nastaveny jako proměnné individuálních strojů nebo jejich skupin. Master, běžné a administrativní uzly jsou určeny přiřazením strojů do příslušných skupin v inventáři. Konfigurace Ansible, jež je součástí řešení, mění pořadí přednosti proměnných, aby proměnné v inventáři měly přednost před proměnnými v playbooku. Díky tomu mohou být proměnné v playbooku používány jako výchozí a inventář může sloužit k úpravě konfigurace. (*Ansible Documentation*, 2022, How to build your inventory, Ansible Configuration Settings)

Ke spuštění playbooku slouží příkaz ansible-playbook, který vypisuje informace o průběhu jeho vykonávání. Pomocí štítků (tag) přiřazených úlohám bylo možné spustit pouze vybranou část nasazení. (*Ansible Documentation*, 2022, ansible-playbook, Tags)

3.2.2 Příprava nasazení

V první fázi bylo potřeba rozhodnout, z jakého počítače bude playbook spouštěn. Mohl být zvolen některý počítač s přístupem k přidělenému stroji, nebo přímo stroj, na kterém bylo operováno. Zvolena byla druhá varianta, protože se v dané situaci jevila výhodnější.

Zvolená konfigurace nasazení se odvíjela od toho, že vytvářené nasazení sloužilo pouze k otestování vyvinuté aplikace a nebylo produkčním. Zmíněny jsou pouze vybrané detaily:

- Protože se Ansible nacházel na stejném stroji, jako na kterém byly vykonávány úlohy, byl stroji v inventáři nastaven plugin pro lokální připojení (*Ansible Documentation*, 2022, ansible.builtin.local connection).
- Stroj byl zařazen do všech tří skupin uzlů clusteru systému pro přepis (master, běžný a administrativní).
- „Vysoká dostupnost“ byla nastavena na nepravdu, aby playbook nenasazoval záložní repliky komponent, které nebyly pro účely nasazení potřebné a pro které neměl stroj dostatečnou výpočetní kapacitu. Tím se nasazení lišilo od nasazení na testovacím clusteru.
- Řada parametrů byla vzhledem k účelu nasazení pro zjednodušení nastavena jen odhadem.

Další konfigurace jsou zmíněny v rámci řešení problémů v následující kapitole.

3.2.3 Provedení nasazení a řešení problémů

Ke spouštění playbooku sloužil již zmíněný příkaz ansible-playbook. Při řešení problémů byl využíván nástroj kubectl (viz 1.2), který umožňoval např.:

- vypsat nasazené pody a jiné objekty v Kubernetes,
- zjistit stav objektů a jejich vlastnosti,
- vypsat log Kubernetes objektu (popisuje např. proč daný pod nemohl být nasazen),
- vypsat log aplikace běžící v podu.

Dále jsou uvedeny vybrané řešené problémy.

Zjištění přístupových údajů k repozitáři s potřebnými artefakty

Součástí konfigurace nasazení je nastavení přístupu k repozitáři, který obsahuje komponenty jádra a služby pro přepis. Ve stejné síti, v jaké byl přidělený stroj, se nacházel také testovací cluster pro systém pro přepis, který měl nasazený repozitář Harbor (viz 1.1) obsahující potřebné artefakty, čehož bylo rozhodnuto využít. Pro přístup k repozitáři ale byly potřeba přístupové údaje.

Z playbooku bylo zjištěno, že přístupové údaje do repozitáře specifikované v inventáři jsou playbookem zapisovány do objektu typu Secret v Kubernetes (viz 1.2.3). Údaje pro přístup do Harboru v testovacím clusteru tak mohly být získány z objektu typu Secret z Kubernetes v testovacím clusteru. Příslušný objekt byl nalezen a jeho obsah vypsán pomocí kubectl, dále bylo jeho obsah nutné dekódovat z kódování base64, ve kterém je obsah objektů Secret zakódován (*Kubernetes Documentation*, ©2022, Secrets).

Zjištění dostupných verzí komponent jádra

Během nasazování bylo zjištěno, že verze komponent jádra systému pro přepis definované v playbooku jako výchozí nebyly v repozitáři obsaženy. Pro nalezení dostupných verzí bylo nahlédnuto do testovacího Kubernetes clusteru, kde bylo možné pomocí příkazu kubectl vypsat popis podů komponent jádra a zjistit názvy a verze používaných obrazů kontejnerů.

Nestartující komponenty Cephu

Jedním z řešených problémů bylo, že nebylo možné nasadit komponenty Cephu. Jejich pody měly, jak bylo možné zjistit z výpisu nástroje kubectl, status „Pending“, tedy že pody nenastartovaly (*Kubernetes Documentation*, ©2022, Pod Lifecycle). Z logu podů v Kubernetes bylo zjištěno, že nasazení brání označení „taint“ s popisem „node-role.kubernetes.io/master:NoSchedule“ na uzlu, které master uzlům automaticky nastavuje nástroj kubeadm (*Kubernetes Documentation*, ©2022, Kubeadm – Implementation details). Bylo tak nutné playbook upravit tak, aby bylo možné pomocí volby v inventáři tento taint volitelně odstranit.

Po této opravě se objevil obdobný problém, tentokrát ale způsobený tím, že bylo opomenuto nastavení speciálního štítku specifického pro systém pro přepis na objekt uzlu. Tento štítek označoval uzly vhodné pro nasazení Cephu a byl komponentami Cephu vyžadován pro jejich nasazení dle nastavení jejich afinity k uzlům.

Neustále se restartující komponenta radosgw

Dalším z problémů bylo, že se komponenta radosgw (Ceph Object Gateway, viz 1.3.4) neustále ihned po svém startu restartovala. Z logu podu ani z logu aplikace v podu se nepodařilo zjistit příčinu. Podezření ale padlo na to, že se ve výpisu podů nenacházela žádná OSD. V playbooku bylo zjištěno, že kvůli chybě v logice playbooku nebyla při zakázané volbě „vysoké dostupnosti“ clusteru žádná OSD nasazována.

Po její opravě a opětovném spuštění playbooku došlo k nasazení OSD a úspěšnému nasazení radosgw.

Nedostupnost Lokiho z Grafany

Posledním popisovaným problémem je, že při stahování logu skončila aplikace pro stahování diagnostických dat s chybou „502 bad gateway“. S pomocí dokumentace (*Grafana Loki documentation*, ©2022b, Troubleshooting) bylo zjištěno, že se Grafana nedokáže připojit k Lokimu. Ve webovém rozhraní Grafany pak bylo zjištěno, že zadaná doména Kubernetes služby, přes kterou se Grafana k Loki připojuje, byla zadána chybně, neboť v adrese chyběla specifikace jmenného prostoru, ve kterém se služba nacházela (viz odkazovaná dokumentace). Po opravě této chyby se dokázala Grafana k Lokimu úspěšně připojit.

3.3 Výsledek nasazení a otestování

Automatizace nasazení byla odladěna do stavu, kdy playbook proběhl bez chyb, vytvořené nasazení neobsahovalo zjevné problémy, bylo možné vytvořit požadavek na přepis a pomocí vyvinuté aplikace o něm bez chyb stáhnout diagnostické údaje.

Problém se vyskytl ve výsledném souboru s grafy metrik, z nichž některé byly prázdné, což indikovalo, že metriky, na které bylo dotazováno, nebyly v Prometheu přítomny. To bylo neočekávané vzhledem k tomu, že při dřívějším testování proti testovacímu clusteru (na kterém byla nasazena jiná verze systému pro přepis) byly všechny metriky přítomny, viz kapitola 2.8.

Ve vzorovém přehledu v Grafaně se požadované metriky rovněž nezobrazovaly a při zkoumání pomocí nástroje Grafana Explore bylo zjištěno, že se požadované metriky popisující pod agenta v systému nevyskytují. Konkrétně se jednalo o metriky s názvy:

- `container_cpu_cfs_periods_total`,
- `container_cpu_cfs_throttled_periods_total`,
- `kube_pod_container_resource_requests`,
- `kube_pod_container_resource_limits`.

Po tomto zjištění bylo po dohodě s vedoucím bakalářské práce testování vůči vlastnímu nasazení ukončeno; bylo prohlášeno, že oprava této chyby v systému pro přepis by byla nad rámec této bakalářské práce.

Závěr

Práce měla tři cíle. Prvním bylo seznámení se se softwarově definovaným úložištěm Ceph a způsobem, jakým je v systému pro přepis řeči na text využíváno. Druhým cílem bylo vyvinutí aplikace, která umožní stahovat uložená diagnostická data o průběhu vykonávání požadavků uživatelů. Posledním cílem bylo zprovoznění privátního nasazení systému pro přepis a otestování běhu vyvinuté aplikace proti němu.

V první části práce, jejíž náplň je teoretická, byl představen stack zahrnující systém Kubernetes, který obstarává správu distribuovaného systému a v rámci něhož jsou všechny další komponenty nasazeny, distribuované úložiště Ceph, jež slouží k dlouhodobému spolehlivému ukládání dat, nástroje Prometheus a Loki, které slouží ke sběru diagnostických dat a které využívají Ceph jako úložiště, a nástroj Grafana, který slouží k vizualizaci dat z Promethea a Lokiho. Tím byl splněn první cíl práce.

Dále byla v rámci práce vyvinuta aplikace, jež na základě zadaného identifikátoru požadavku na přepis stáhne metriky a log popisující jeho průběh a zpracuje je do výstupního souboru vhodného pro analýzu člověkem. V rámci jejího vývoje bylo třeba mimo jiné navrhnout vhodnou interpretaci informací o požadavku, implementovat komunikaci s rozhraními nástrojů Prometheus a Loki a vyřešit jejich omezení a navrhnout dotazy, které získají požadovaná data ve vhodné podobě. Výsledná implementace fungovala vůči systému na testovacím clusteru za testovaných okolností korektně. Vzniklé řešení ukazuje, jakým způsobem lze vybudovat vlastní diagnostické nástroje pracující s těmito technologiemi.

Na závěr práce byla pomocí poskytnutého automatizovaného řešení využívajícího nástroj Ansible nasazena na poskytnutý stroj privátní instance systému pro přepis, proti níž byla vyvinutá aplikace otestována. V rámci této části byla opravena řada chyb v automatizaci nasazení, přičemž byly využity zejména znalosti o platformě Kubernetes. Chování vyvinuté aplikace bylo i proti tomuto nasazení shledáno korektním. Bylo ale odhaleno, že v nasazeném systému chyběly některé metriky; oprava tohoto problému byla shledána jako sahající za rámec této práce. Vyvinutím aplikace a jejím otestováním proti privátnímu nasazení byl splněn druhý a třetí cíl práce.

V budoucnu by bylo vhodné provést podrobnější integrační testy pro vyvinutou aplikaci, které v rámci této práce nebyly vyžadovány.

Bylo shledáno, že vývoj proti privátnímu prostředí by byl výhodnější, protože by nedocházelo k neočekávaným změnám v systému pro přepis, které vývoj narušovaly. Vlastní prostředí je nicméně náročné na nasazení a správu.

Seznam použité literatury

Amazon S3, ©2022 [online]. Amazon Web Services, Inc. or its affiliates. [cit. 2022-06-27]. Dostupné z: https://aws.amazon.com/s3/.

Ansible Documentation, 2022 [online]. Ansible project contributors. Verze 5 [cit. 2022-09-15]. Dostupné z: https://docs.ansible.com/ansible/5/index.html.

Apache ECharts, ©2017–2022 [online]. The Apache Software Foundation. [cit. 2022-06-08]. Dostupné z: https://echarts.apache.org/en/index.html. Verze 5.3.2.

archive/tar, 2022. In: *Go Packages* [online]. Verze 1.17.11 [cit. 2022-06-10]. Dostupné z: https://pkg.go.dev/archive/tar@go1.17.11.

archive/zip, 2022. In: *Go Packages* [online]. Verze 1.17.11 [cit. 2022-06-10]. Dostupné z: https://pkg.go.dev/archive/zip@go1.17.11.

BRAZIL, Brian, 2016. Exposing the software version to Prometheus. *Reliable Insights* [online]. Robust Perception [cit. 2022-09-14]. Dostupné z: https://www.robustperception.io/exposing-the-software-version-to-prometheus/.

Ceph Documentation, ©2016 [b.r.] [online]. Ceph authors and contributors. Verze pacific [cit. 2022-05-08]. Dostupné z: https://docs.ceph.com/en/pacific/.

Cloud Native Computing Foundation, ©2022 [online]. The Linux Foundation. [cit. 2022-06-16]. Dostupné z: https://www.cncf.io/.

Documentation - The Go Programming Language, [b.r.] [online]. Google. [cit. 2022-09-12]. Dostupné z: https://go.dev/doc.

etcd, ©2013–2022 [online]. etcd Authors. [cit. 2022-06-11]. Dostupné z: https://etcd.io/.

github.com/spf13/cobra, 2022. In: *Go Packages* [online]. Verze 1.4.0 [cit. 2022-06-09]. Dostupné z: https://pkg.go.dev/github.com/spf13/cobra@v1.4.0.

github.com/spf13/viper, 2021. In: *Go Packages* [online]. Verze 1.10.1 [cit. 2022-06-09]. Dostupné z: https://pkg.go.dev/github.com/spf13/viper@v1.10.1.

go-echarts/go-echarts, 2022. In: *GitHub* [online]. Commit 59126fe [cit. 2022-06-08]. Dostupné z: https://github.com/go-echarts/go-echarts/tree/59126fe22eac4.

google/cadvisor, 2022. In: *GitHub* [online] [cit. 2022-05-05]. Dostupné z: https://github.com/google/cadvisor.

Grafana documentation, ©2022 [online]. Grafana Labs. Verze 8.3 [cit. 2022-06-15]. Dostupné z: https://grafana.com/docs/grafana/v8.3/.

grafana/loki, 2022. In: *GitHub* [online]. Verze 2.4.2 [cit. 2022-06-12]. Dostupné z: https://github.com/grafana/loki/tree/v2.4.2.

Grafana Loki documentation, ©2022a [online]. Grafana Labs. Verze 2.4.x [cit. 2022-08-04]. Dostupné z: https://grafana.com/docs/loki/v2.4.x/.

Grafana Loki documentation, ©2022b [online]. Grafana Labs. Verze 2.5.x [cit. 2022-09-12]. Dostupné z: https://grafana.com/docs/loki/v2.5.x/.

Harbor, ©2022 [online]. Harbor Authors & The Linux Foundation. [cit. 2022-06-16]. Dostupné z: https://goharbor.io.

Helm Docs, ©2022 [online]. Helm Authors & The Linux Foundation. [cit. 2022-06-19]. Dostupné z: https://helm.sh/docs/.

How Ansible Works, ©2020. *Ansible* [online]. Red Hat, Inc. [cit. 2022-06-03]. Dostupné z: https://www.ansible.com/overview/how-ansible-works.

html/template, 2022. In: *Go Packages* [online]. Verze 1.17.11 [cit. 2022-06-08]. Dostupné z: https://pkg.go.dev/html/template@go1.17.11.

Kubernetes Documentation, ©2022 [online]. The Kubernetes Authors & The Linux Foundation. Verze 1.23 [cit. 2022-06-17]. Dostupné z: https://v1-23.docs.kubernetes.io/docs/home/.

kubernetes/kube-state-metrics, 2022. In: *GitHub* [online] [cit. 2022-05-02]. Dostupné z: https://github.com/kubernetes/kube-state-metrics.

kubernetes/kubernetes, 2021. In: *GitHub* [online]. Verze 1.23.0 [cit. 2022-09-12]. Dostupné z: https://github.com/kubernetes/kubernetes/tree/v1.23.0.

MAAS Documentation, ©2022 [online]. Canonical Ltd. [cit. 2022-05-30]. Dostupné z: https://maas.io/docs. MAAS verze 3.1.

MARCHBANKS, Chris, 2019. *Containing your Cardinality* [online]. SPLUNK INC. [cit. 2022-06-12]. Dostupné z: https://promcon.io/2019-munich/slides/containing-your-cardinality.pdf.

prometheus/client-go, 2021. In: *GitHub* [online]. Verze 1.11.0 [cit. 2022-09-12]. Dostupné z: https://github.com/prometheus/client_golang/tree/v1.11.0.

Prometheus Docs, ©2014–2022 [online]. Prometheus Authors & The Linux Foundation [cit. 2022-06-19]. Dostupné z: https://prometheus.io/docs/introduction/overview/. Prometheus verze 2.33.

prometheus/prometheus, 2022. In: *GitHub* [online]. Verze 2.33.5 [cit. 2022-06-12]. Dostupné z: https://github.com/prometheus/prometheus/tree/v2.33.5.

prometheus-community/helm-charts, 2021. In: *GitHub* [online]. Verze kube-prometheus-stack-33.2.1 [cit. 2022-09-12]. Dostupné z: https://github.com/prometheus-community/helm-charts/tree/kube-prometheus-stack-33.2.1.

Protocol Buffers, [b.r.] [online]. Google. [cit. 2022-06-12]. Dostupné z: https://developers.google.com/protocol-buffers.

Rook Ceph Documentation, ©2022 [online]. Rook Authors & The Linux Foundation. Verze 1.8 [cit. 2022-06-21]. Dostupné z: https://rook.github.io/docs/rook/v1.8/.

SPALETA, Jef, 2019. How Kubernetes works. *Cloud Native Computing Foundation Blog* [online] [cit. 2022-05-10]. Dostupné z: https://www.cncf.io/blog/2019/08/19/how-kubernetes-works/.

thanos-io/thanos, 2021. In: *GitHub* [online]. Verze 0.24.0 [cit. 2022-09-12]. Dostupné z: https://github.com/thanos-io/thanos/tree/v0.24.0.

time, 2022. In: *Go Packages* [online]. Verze 1.17.11 [cit. 2022-09-13]. Dostupné z: https://pkg.go.dev/time@go1.17.11.

WEIL, Sage, 2015 [2015-09-30T09:01:52-07:00]. *Re: chain replication scheme* [e-mailová komunikace]. [cit. 2022-06-17]. Dostupné z: http://lists.ceph.com/pipermail/ceph-users-ceph.com/2015-September/005060.html.

WEIL, Sage, 2019. Intro to Ceph. In: *YouTube* [online] [cit. 2022-06-17]. Dostupné z: https://www.youtube.com/watch?v=PmLPbrf-x9g. Kanál uživatele Ceph.

WEIL, Sage A., Scott A. BRANDT, Ethan L. MILLER et al., 2006. *CRUSH: Controlled, Scalable, Decentralized Placement of Replicated Data* [online]. [cit. 2022-02-06]. University of California, Santa Cruz. Dostupné z: https://ceph.com/assets/pdfs/weil-crush-sc06.pdf.

WEIL, Sage A., Andrew W. LEUNG, Scott A. BRANDT et al., 2007. *RADOS: A Scalable, Reliable Storage Service for Petabyte-scale Storage Clusters* [online]. [cit. 2021-12-28]. University of California, Santa Cruz. Dostupné z: https://ceph.com/assets/pdfs/weil-rados-pdsw07.pdf.