VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
BRNO UNIVERSITY OF TECHNOLOGY

FAKULTA INFORMAČNÍCH TECHNOLOGIÍ
FACULTY OF INFORMATION TECHNOLOGY

ÚSTAV INFORMAČNÍCH SYSTÉMŮ
DEPARTMENT OF INFORMATION SYSTEMS

POUŽITÍ KONCEPTU OPEN TELEMETRY PRO SPRÁVU SÍTĚ

USING OPEN TELEMETRY FOR NETWORK MONITORING AND MANAGEMENT

BAKALÁŘSKÁ PRÁCE
BACHELOR'S THESIS

AUTOR PRÁCE **ŠTĚPÁN BAKAJ**
AUTHOR

VEDOUCÍ PRÁCE **doc. Ing. PETR MATOUŠEK, Ph.D., M.A.**
SUPERVISOR

BRNO 2023

VYSOKÉ UČENÍ TECHNICKÉ V BRNĚ
FAKULTA INFORMAČNÍCH TECHNOLOGIÍ

Zadání bakalářské práce

148611

Ústav: Ústav informačních systémů (UIFS)
Student: **Bakaj Štěpán**
Program: Informační technologie
Specializace: Informační technologie

Název: **Použití konceptu Open Telemetry pro správu sítě**
Kategorie: Počítačové sítě
Akademický rok: 2022/23

Zadání:

1. Seznamte se s konceptem Open Telemetry pro správu sítí. Popište architekturu systému Open Telemetry včetně způsobu sběru monitorovaných dat a jejich přenosu.
2. Proveďte rešerši dostupných nástrojů pro Open Telemetry a podporu tohoto konceptu u síťových zařízení a služeb.
3. Navrhněte testovací prostředí pro implementaci konceptu Open Telemetry. Popište, jaké informace budete monitorovat a jak je lze zpracovávat. Implementujte testovací prostředí.
4. Proveďte dlouhodobé monitorování testovací sítě pomocí konceptu Open Telemetry. Uvažujte různé situace v testovacím prostředí (výpadek, zahlcení, kybernetický útok). Analyzujte monitorovací data a ukažte, jak je lze využít k detekci incidentů na síti.
5. Srovnejte možnosti konceptu Open Telemetry vůči monitorování toků (NetFlow), sběru dat SNMP a logování událostí syslog. Ukažte přednosti a omezení systému Open Telemetry vůči stávajícím konceptům správy sítě.
6. Zhodnoťte přínos své práce a možnosti praktického využití.

Literatura:

- Open Telemetry Specification. Dokumentace dostupná na URL https://opentelemetry.io/docs/reference/specification/ (září 2022).
- Open Telemetry Protocol (OTLP). Dokumentace dostupná na URL https://opentelemetry.io/docs/reference/specification/protocol/ (září 2022).
- Benoit Claise and Ralf Wolter. Network Management: Accounting and Performance Strategies. Cisco Press, 2007.

Při obhajobě semestrální části projektu je požadováno:

- Body 1-3.

Podrobné závazné pokyny pro vypracování práce viz https://www.fit.vut.cz/study/theses/

Vedoucí práce: **Matoušek Petr, doc. Ing., Ph.D., M.A.**
Vedoucí ústavu: Kolář Dušan, doc. Dr. Ing.
Datum zadání: 1.11.2022
Termín pro odevzdání: 10.5.2023
Datum schválení: 18.10.2022

Abstrakt

Tato bakalářská práce se zabývá novým systémem OpenTelemetry, jež se využívá pro sběr telemetrických dat. Cílem práce je ukázat, jak systém funguje a jaké telemetrické data nám poskytuje. Pomocí experimentů, jsme si ukázali, jaké data je nám schopen systém poskytnout a jaké nežadoucí stavy jsme schopni detekovat. Systém jsme porovnali s aktuálně využívanými nástroji.

Abstract

This bachelor thesis deals with the new OpenTelemetry system, which is used for telemetry data collection. The aim of the thesis is to show how the system works and what telemetry data it provides. By means of experiments, we have shown what data the system is able to provide us and what unwanted states we are able to detect. We compared the system with currently used tools.

Klíčová slova

OpenTelemetry, telemetrická data, monitorování, stopy, záznamy, metriky, kolektor

Keywords

OpenTelemetry, telemetry data, monitoring, traces, logs, metrics, collector

Citace

BAKAJ, Štěpán. *Použití konceptu Open Telemetry pro správu sítě*. Brno, 2023. Bakalářská práce. Vysoké učení technické v Brně, Fakulta informačních technologií. Vedoucí práce doc. Ing. Petr Matoušek, Ph.D., M.A.

Použití konceptu Open Telemetry pro správu sítě

Prohlášení

Prohlašuji, že jsem tuto bakalářskou práci vypracoval samostatně pod vedením pana doc. Ing. Petra Matouška, Ph.D., M.A. Uvedl jsem všechny literární prameny, publikace a další zdroje, ze kterých jsem čerpal.

.....
Štěpán Bakaj
8. května 2023

Poděkování

Rád bych poděkoval doc. Ing. Petru Matouškovi, Ph.D., M.A. za jeho ochotu, vstřícnost a trpělivé vedení mé práce a rodičům za podporu během celé doby studia.

Obsah

Seznam obrázků

Slovník pojmů

API aplikačně programové vybavení, které v systému OpenTelemetry definuje, jak budou telemetrická data generována.

Baggage představují sdílená data stopy.

Exportér (Exporter) část kolektoru, jež je zodpovědná za odesílání telemetrických dat z kolektoru do monitorovacích nástrojů.

Kolektor (Collector) je komponenta, která přijímá a odesílá telemetrická data.

Operace (Span) představitel jedné operace ve stopě.

Procesor část kolektoru, která dokáže filtrovat, upravovat a mazat přijaté záznamy na kolektor.

Protobuf celým názvem Protocol Buffers Protobuf, jedná se o mechanizmus serializace dat ve strukturované podobě.

Přijímač (Receiever) část kolektoru, která je zodpovědná za příjem telemetrickým dat.

Stopy (Traces) nám dávají přehled, co se děje při provádění jedné nebo více operací, které spolu souvisí.

Telemetrická data v systému OpenTelemetry mluvíme o záznamech, metrikách a stopách.

Záznam (Log) popisuje událost, která se odehrála a kde se odehrála s časovým razítkem.

Kapitola 1

Úvod

Obsahem této práce je porovnání nového standardu pro sběr telemetrických dat. Toto téma se může zdát vcelku jednoduché, ale po bližším prozkoumání narazíme na spousty problémů.

Neexistuje univerzální nástroj, jež by umožňoval sběr všech telemetrických dat (záznamy, metriky a stopy) z aplikací napsaných pomocí různých technologií. Právě toho chce dosáhnout nový systém OpenTelemetry, který nadefinoval standard pro formát telemetrických dat a vytvořil knihovny pro automatické nebo manuální generování telemetrických dat. Díky tomu není problém měnit monitorovací nástroje nebo použít více nástrojů najednou.

Cílem této práce je porovnat nynější řešení se systémem OpenTelemetry. Provést dlouhodobé monitorování a zkoumat jak budou telemetrická data vizualizována, když dojde například k výpadku služby nebo kybernetickému útoku. Analyzovat telemetrická data a ukázat, jak je lze využít k detekci incidentů na síti.

Tato práce může pomoct, při rozhodování o využití systému OpenTelemetry. Čtenář se dozví, jaká telemetrická data nám systém poskytne, jeho výhody a nevýhody. Ukážeme si jaké chyby pomocí něj dokážeme detekovat a pomocí čeho si nasbíraná data vizualizovat.

V další kapitole si povíme o systému OpenTelemetry, jaká telemetrická data sbíráme a pomocí jakých protokolů komunikujeme mezi prvky v systému. Třetí kapitola obsahuje popis aplikace, na které budeme provádět testování a jaké monitorovací nástroje využíváme a co s nimi sledujeme. Ve čtvrté kapitole se podíváme na porovnání s konkurenčními nástroji a popíšeme si, jaké přednosti či omezení systém OpenTelemetry má.

Kapitola 2

Monitorovací systém OpenTelemetry

OpenTelemetry vzniklo spojením dvou projektů OpenTracing a OpenCensus. Jeho název se skládá ze dvou slov open, které značí, že se jedná o otevřený standard a telemetry. Slovo telemetry je složenina dvou řeckých slov *tele*, neboli vzdálený, a *metry*, což znamená v překladu měření. Byl vydán v květnu roku 2019.

Je to nástroj s otevřeným zdrojovým kódem pro pozorování aplikací. Nabízí rozhraní, sady pro vývoj softwaru (SDK) a další nástroje pro sběr telemetrických dat z aplikací a hardwaru, na kterém aplikace běží, pro lepší porozumění jejich stavu a výkonu.

Než přišla OpenTelemetry, bylo velmi těžké přecházet mezi monitorovacími nástroji, protože každý nástroj měl vlastní knihovny a agenty pro sběr a vysílání dat. Proto přišli se standardizovaným formátem dat pro odesílání do monitorovacího nástroje, jako je například Jaeger[1] nebo Prometheus[2]. Pokud se uživatel po nějaké době rozhodne změnit monitorovací nástroj, tak nemusí měnit knihovnu pro sběr telemetrických dat a nastavovat nové agenty, aby bylo možné posílat telemetrická data do jiného nástroje.

2.1 Vlastnosti systému OpenTelemetry

Systém OpenTelemetry má velkou podporu u poskytovatelů cloudových služeb, prodejců a koncových zákazníků. Hlavní výhody použití:

- jedna knihovna pro programovací jazyky[3] s podporou automatického a manuálního generování telemetrických dat
- jediný kolektor, který můžeme nasadit lokálně, v cloudu nebo jako kontejner
- úplná kontrola nad daty a podpora paralelního odesílání do více nástrojů pomocí konfigurace
- schopnost šíření dat paralelně v různých formátech

OpenTelemetry není monitorovací nástroj jako již zmíněný Jaeger nebo Prometheus. Místo toho nám sbírá a přenáší data, o kterých si povíme v další podkapitole.

[1] https://www.jaegertracing.io/ [21.11.2022]
[2] https://prometheus.io/ [21.11.2022]
[3] Podporované programovací jazyky https://opentelemetry.io/docs/instrumentation/ [21.11.2022]

2.2 Monitorovaná data

Pomocí OpenTelemetry sbíráme monitorovací data zvané *signály*. Je to skupina telemetrických dat podporovaná specifikací systému OpenTelemetry. Nyní podporuje tyto signály:

- **Stopy (Traces)** představují jednu nebo více operací
- **Metriky** jsou naměřené numerické hodnoty
- **Záznamy (Logs)** jsou události zaznamenané s časovou značkou
- **Sdílené data (Baggage)** slouží pro výměnu dat mezi operacemi

2.2.1 Sbírání stop (Tracing)

Stopy [12] nám dají přehled o tom, co se stane, když uživatel nebo aplikace vyvolá požadavek.

```
{
    "name": "Greetings"  # název operace, kterou představuje
    "context": {
        # id stopy, do které patří
        "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
        # id aktuální operace
        "span_id": "0x5fb397be34d26b51",
    },
    "parent_id": null,
    "start_time": "2022-04-29T18:52:58.114304Z",
    "end_time": "2022-04-29T18:52:58.114435Z",
    # atributy operace
    "attributes": {
        "http.route": "some_route1"
    },
    # události operace
    "events": [
        {
            "name": "hey there!",
            "timestamp": "2022-04-29T18:52:58.114561Z",
            "attributes": {
                "event_attributes": 1
            }
        }
    ],
}
```

Výpis 1: Ukázka stopy s jednou operací

Na výpisu 1 vidíme jednoduchou stopu s jednou operací, který se jmenuje `Greetings`.

Pro lepší pochopení stop v systému OpenTelemetry, bychom si měli nejprve popsat, jak stopy vznikají a položky stop, které hrají velkou roli při psaní kódu pro sběr telemetrických dat.

- **Poskytovatel stop (Trace Provider)**

  Poskytovatel stop si můžeme představit jako továrnu na trasy. Skoro ve všech aplikacích se inicializuje pouze jednou a žije celou dobu běhu aplikace. Typicky se jedná o první věc, kterou programátor udělá, pokud chce trasovat pomocí OpenTelemetrie. U některých programovacích jazyků je definován globálně za nás.

- **Tracer**

  Stopy se skládají z jedné nebo více operací. Operace obsahuje informaci o sobě a právě tracer nám poskytuje monitorovací data o operaci.

- **Exportér stop (Trace Exporters)**

  Exportér stop odesílá stopy na předem definovaná místa. Těmi mohou být kolektor, standardní výstup pro ladění kódu nebo monitorovací nástroj.

- **Kontext stopy**

  Kontextem stopy rozumíme metadata o operacích, která zajišťují souvislosti mezi operacemi celou aplikací. Představme si, že služba A volá službu B a chceme tuto událost sledovat pomocí trasy. V tom případě se použije kontext stopy k zachycení ID operace a trasy služby A, aby se při vytvoření operace služby B mohly spojit operace do trasy. Tento postup nazýváme propagací kontextu.

- **Propagace kontextu**

  Propagace kontextu je základní princip, který umožňuje distribuované trasovaní. Právě díky tomuto principu mohou být monitorovací data o operaci, která jsou generována jinde, spojeny do jedné trasy. Tento princip se skládá ze dvou částí.

  - **Kontext**

    Kontextem rozumíme objekt, který nese informace k odesílající a přijímající službě a ty umožňují spojit jednu operaci s druhou a přidat je do trasy. Kontext můžeme vidět na obrázku 1, kde se nachází pod klíčem `context`.

  - **Propagace**

    Propagací nebo šířením nazýváme mechanismus, který přenáší monitorovací data o operaci mezi službami a procesy a skládá tím distribuovanou stopy.

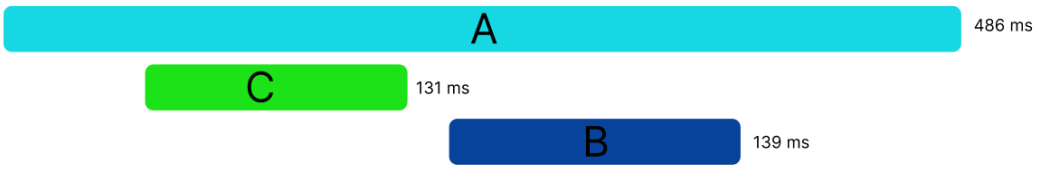
Pro lepší představu si tu ukážeme, jak by vypadala data distribuované trasy v JSON[4] formátu a jak by nám je mohl zobrazit monitorovací nástroj.

[4] JavaScript Object Notation – způsob zápisu dat

```
# podoperace
{
    "name": "C", # název operace
    "context": {
        # identifikátor stopy, které operace náleží
        "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
        # identifikátor operace
        "span_id": "0x5fb397be34d26b51",
    },
    # identifikátor rodičovské operace
    "parent_id": "0x051581bf3cb55c13",
    "start_time": "2022-04-29T18:52:58.114304Z", # začátek operace
    "end_time": "2022-04-29T18:52:58.114435Z",  # konec operace
}
# podoperace
{
    "name": "B",
    "context": {
        "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
        "span_id": "0x93564f51e1abe1c2",
    },
    "parent_id": "0x051581bf3cb55c13",
    "start_time": "2022-04-29T18:52:58.114492Z",
    "end_time": "2022-04-29T18:52:58.114631Z",
}
# hlavní operace
{
    "name": "A",
    "context": {
        "trace_id": "0x5b8aa5a2d2c872e8321cf37308d69df2",
        "span_id": "0x051581bf3cb55c13",
    },
    "parent_id": null,
    "start_time": "2022-04-29T18:52:58.114201Z",
    "end_time": "2022-04-29T18:52:58.114687Z",
}
```

Výpis 2: Ukázka distribuované stopy ve formátu JSON

Na výpisu 2 vidíme tři operace A, B a C, které obsahují kontext s `trace_id` a `span_id`. `Trace_id` nám říká, pod jakou trasu operace spadá a `span_id` reprezentuje identifikátor dané operace. Pomocí `parent_id` spojujeme jednotlivé operace do jedné distribuované stopy. Tento soubor by nám monitorovací nástroj vykreslit právě takto:



Obrázek 2.1: Ukázka distribuované stopy

Na obrázku 2.1 vidíme jednu hlavní operaci s názvem A, která obsahuje dvě podoperace B a C. Když nám monitorovací nástroj spojí operace dohromady, tak je nazýváme distribuované stopy.

Monitorovací data o operaci (Span)

Operace reprezentují hlavní jednotku práce ve stopě. Operace v systému OpenTelemetry se skládá z následujících položek:

- jména
- rodičovského ID operace – pro kořen je prázdný
- začáteční a koncový čas
- kontextu
- atributů
- událostí
- spojení
- statutu

Na výpisu 3 se můžeme podívat na ukázkové monitorovací data operace.

```
{
  # identifikátor stopy, které operace náleží
  "trace_id": "7bba9f33312b3dbb8b2c2c62bb7abe2d",
  "parent_id": "",
  # identifikátor operace
  "span_id": "086e83747d0e381e",
  # název operace, který v~tomto případě nese cestu koncového bodu
  "name": "/v1/sys/health",
  "start_time": "2021-10-22 16:04:01.209458162 +0000 UTC",
  "end_time": "2021-10-22 16:04:01.209514132 +0000 UTC",
  "status_code": "STATUS_CODE_OK",
  # atributy operace
  "attributes": {
    "net.transport": "IP.TCP", # využitý protokol pro přenos
    "net.peer.ip": "172.17.0.1", # vzdálená IP adresa
    "net.peer.port": "51820", # vzdálený port
    "net.host.ip": "10.177.2.152", # lokální IP adresa
    "net.host.port": "26040", # lokální port
    "http.method": "GET", # HTTP metoda požadavku
    "http.target": "/v1/sys/health", /* cesta koncového bodu
    "http.scheme": "http"  # využitý protokol
  },
  # pole událostí operace
  "events": [
    {
      "message": "OK",
      "timestamp": "2021-10-22 16:04:01.209512872 +0000 UTC"
    }
  ]
}
```

Výpis 3: Ukázka operace ve formátu JSON představující HTTP GET požadavek

Operace mohou být vnořené pomocí rodičovského ID operace. Tím dosáhneme lepšího porozumění toho, co se děje při provádění operace, která podoperace nás nejvíce zdržuje a co se během ní děje. Dále si popíšeme jednotlivé parametry operací a jejich význam.

- **Kontext**

  Kontext je neměnný soubor dat pro danou operaci a skládá se z ID stopy, které náleží, unikátního klíče operace, příznaků stopy a stavu stopy. Stav je seznam atributů ve tvaru klíč-hodnota, jež nesou specifické informace pro daný nástroj.

- **Atributy**

  Atributy nesou data ve formátu klíč-hodnota, které obsahují metadata, pomocí nichž můžeme charakterizovat reálnou operaci, kterou mají monitorovací data představovat.

Atribut musí splňovat následující pravidla platící pro každý programovací jazyk. Klíč musí být nenulový řetězec a hodnota může nabývat pouze nenulový řetězec, logickou hodnotou, desetinné číslo, celé číslo nebo pole těchto hodnot.

- **Události**

  Události si můžeme představit jako strukturovanou zprávu operace, jež má poukázat na významný okamžik v průběhu operace. Pro lepší pochopení si to vysvětlíme na příkladu. Sledujeme načítaní dat ze serveru, takže vhodnou událostí by bylo, od kdy data byla dostupná.

- **Spojení**

  Spojení představuje propojení jedné operace s další nebo více operacemi. Řekněme, že máme distribuovaný systém, v němž sledujeme jeho operace pomocí stop.

  Při provedení některé z operací se může jiná operace ocitnout ve frontě, kde čeká na provedení, jejíž provedení je však asynchronní. Obě operace sledujeme pomocí stop. Rádi bychom spojili stopu operace z fronty s hlavní operací, ale nemůžeme předvídat, kdy se operace spustí. Proto použijeme spojení na jejich propojení. Provedeme to tím, že poslední operace z první stopy propojíme s první operací z druhé stopy.

- **Status**

  K operaci přidáváme status pouze tehdy, když v aplikaci nastane známá chyba, například výjimka. Může obsahovat jedno z následujících označení:

  - Unset
  - Ok
  - Error

  Když výjimku zpracujeme, může být stav operace nastaven na `Error`. V opačném případě je status nastaven na `Unset`. Nastavením stavu na `Unset` může monitorovací nástroj, který zpracovává telemetrická data operace, přiřadit konečný stav.

2.2.2 Metriky

Metrická data [10] měříme a zachytáváme za běhu služby. Okamžik, kdy dojde k zachycení dat, nazýváme metrická událost. Metrika se neskládá pouze z dat měření, ale obsahuje i čas, kdy bylo měření provedeno a související metadata.

Jsou důležitým ukazatelem dostupnosti a vytíženosti aplikací a požadavků na rozhraní. Definováním vlastních metrických dat můžeme zjistit, jak dostupnost naší aplikace ovlivňuje například její návštěvnost. Nasbíraná data nás mohou upozornit na výpadek služby nebo můžeme podle nastavených pravidel automaticky škálovat aplikaci podle poptávky.

Systém OpenTelemetry definuje tři nástroje pro práci s metrickými daty:

- **Počítadlo (Counter)**

  Počítadlo představuje hodnotu, která se v čase sčítá a roste pouze nahoru. Můžeme si jej představit jako počet zápisů na disk.

- **Součet (Measure)**

  Součet slouží pro sčítání hodnot v čase a je definovaný v určitém rozsahu. Můžeme si ho představit jako počet ujetých kilometrů na tachometru.

- **Pozorovatel (Observer)**

  Pozorovatel zachycuje aktuální hodnoty v určitou časovou dobu, například aktuální rychlost auta.

Dalším důležitým pojmem, se kterým se setkáváme u metrických dat je agregace. Agregace je metoda, při které vezmeme velký počet naměřených vzorků a spojíme je do přesných nebo odhadovaných statistik o metrických událostech, jež se odehráli v určitém časovém rozmezí. Systém OpenTelemetry nám neumožňuje tyto agregace vytvářet, ale poskytuje nám běžně agregace, které podporují monitorovací nástroje a vizualizéry jako je suma, počet, poslední hodnota a histogramy.

Metriky jsou určeny k poskytování souhrnných statistických dat. Zde si ukážeme na co jsou vhodné:

- využití procesoru nebo paměti
- počet požadavků na rozhraní
- měření latence operací
- hlášení aktuálně aktivních zpracovávaných požadavků

2.2.3 Záznamy (Logs)

Z uvedených telemetrických dat jsou záznamy [9] pravděpodobně nejznámějším a nejstarším způsobem sběru dat ze služeb nebo aplikací. Většina programovacích jazyků má podporu sběru záznamů již v sobě zabudovanou nebo existuje knihovna pro daný jazyk. V čem spočívá výhoda sběru záznamu v systému OpenTelemetry si popíšeme později v této kapitole.

Záznam uchováváme v podobě textového zápisu s časem, kdy byl záznam zachycen. Může být strukturovaný, což je doporučeno i v dokumentaci, nebo v nestrukturované podobě s metadaty. Záznamy můžeme uchovávat samostatně, jako nezávislý zdroj dat, ale mohou být součástí operace. V systému OpenTelemtry považujeme všechna data za záznamy, pokud nejsou obsažena v distribuované stopě nebo metrice. Záznamy se často používají při pádu aplikace. Právě v nich hledáme hlavní příčinu pádu a při jaké změně k pádu došlo, ale to je jen jedna ukázka použití záznamu. Existuje jich mnoho typů, jako například:

- **Záznam z aplikace**

  Záznam z aplikace nese informaci o události, jež se vyskytla za běhu aplikace.

- **Systémové záznamy**

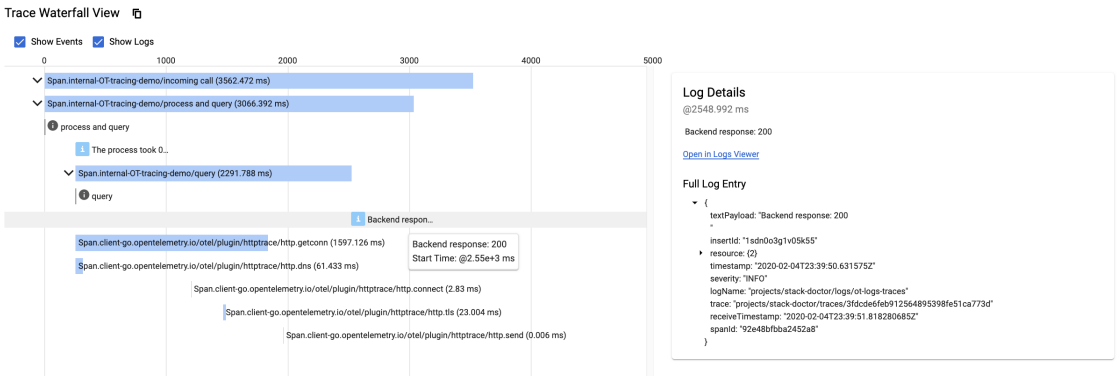
  Systémové záznamy obsahují informaci o samotném operačním systému, kde služba nebo aplikace běží.

- **Síťové záznamy**

Síťové záznamy nám poskytují informace o zařízeních v infrastruktuře na základě aktivity v síti.

- **Záznamy z webových serverů**

  Záznamy z webových serverů, jako je například Apache[5], lze použít pro identifikaci úzkých míst výkonu.



Obrázek 2.2: Záznam připojený k operaci[6]

Na obrázku vidíme záznam, který nese textovou informaci v podobě úspěšné odpovědi ze serveru, časovou značku kdy byl vytvořen a přijat monitorovacím nástrojem, název záznamu, identifikátor stopy a operace, které náleží.

Výhody sběru záznamu se systémem OpenTelemetry

Velké monitorovací nástroje by měly snadno umožňovat propojení telemetrických dat. Většina uživatelů, jež se snaží sbírat telemetrická data, používá pro každý telemetrický signál jiný nástroj a nemá možnost tyto signály propojit. Aktuální řešení sběru záznamu nepodporují propojení s ostatními signály. Cílem systému OpenTelemetry je shromažďovat záznamy s kontextem, aby mohly být později propojeny s dalšími telemetrickými signály. [2]

Struktura záznamu/události systému OpenTelemetry

Systém OpenTelemetry nedělá rozdíl mezi záznamem a událostí z pohledu struktury dat. Pro obojí se využívá stejná struktura. Rozlišujeme je podle dat, která představují. Chcemeli zaznamenat pouze událost v určitý čas, hovoříme o události (výjimka v kódu). Pokud potřebujeme zaznamenat další dodatečné informace o události, poté mluvíme o záznamem.

Struktura záznamu/události [7] byla navržen tak, aby umožňoval reprezentovat záznamy z různých zdrojů. Díky tomu lze mapovat většinu záznamů třetích stran na strukturu záznamu systému OpenTelemetry.

[5] Webový server Apache https://apache.org/ [25.11.2022]

[6] Obrázek byl převzat z webu https://medium.com/google-cloud/integrating-tracing-and-logging-with-opentelemetry-and-stackdriver-a5396fbc3e78 [25.11.2022]

Název	Formát	Popis
Časové razítko	uint64	Čas, kdy událost nastala.
Časové razítko záznamu	uint64	Čas, kdy byla událost zaznamenána.
ID stopy	sekvence bajtů	Identifikátor stopy
ID operace	sekvence bajtů	Identifikátor operace
Příznak stopy	byte	W3C příznak stopy
Text závažnosti	řetězec	Známe také jako úroveň záznamu
Číslo závažnosti	number	Číselná hodnota úrovně závažnosti
Tělo	řetězec	Tělo záznamu/události
Zdroj	map<řetězec, hodnota>	Popis zdroje záznamu/události
Místo generování	tuple s řetězci	Místo odkud byl záznam vygenerován
Atributy	map<řetězec, hodnota>	Dodatečné informace o záznamu-/události

Tabulka 2.1: Struktura záznamu/události systému OpenTelemetry

- **Časové razítko (Timestamp)**

  Čas je udaný v nanosekundách od počátku unixového času. Hodnota je zaznamenána v době, kdy událost nastala, a používá se čas zdroje. Pole je nepovinné.

- **Časové razítko záznamu (ObservedTimestamp)**

  Čas je udaný v nanosekundách od počátku unixového času. Hodnota je zaznamenána při generování záznamu/události systémem OpenTelemetry, potom se toto pole rovná `Timestamp`. Pokud se jedná o záznam, který pochází od třetí strany, tak se jedná o čas, kdy systém OpenTelemetry (kolektor) přijal daný záznam. Pole je povinné.

- **Pole pro kontext stopy**

  - **ID stopy (TraceId)**

    Pole je nastaveno, pokud je záznam/událost součástí stopy. Pole je nepovinné.

  - **ID operace (SpanId)**

    Pole je nastaveno, pokud je záznam/událost součástí operace. Pole je nepovinné.

  - **Příznak stopy (TraceFlags)**

    Poje je definováno specifikací W3C[7] a nyní podporuje akorát jeden příznak `SAMPLED`. Pole je nepovinné.

- **Pole závažnosti**

  - **Text závažnosti (SeverityText)**

    Jedná se o původní textovou reprezentaci závažnosti. Známou taky jako úroveň závažnosti. Pole je nepovinné.

[7] Odkaz na W3C specifikaci. https://www.w3.org/TR/trace-context/#trace-flags [14.03.2023]

- **Číslo závažnosti (SeverityNumber)**

  Jedná se o číselnou hodnotu, která je definovaná specifikací systému OpenTelemetry. Pole je nepovinné.

- **Tělo (Body)**

  Hodnota je obsažena v tělu záznamu/události, zpráva může být čitelná pro lidi nebo to mohou být strukturovaná data. Pole je nepovinné.

- **Zdroj (Resource)**

  Popisujeme zdroj záznamu/události. Můžeme zde popsat, jaká aplikace záznam vytvořila a na jakém stroji běží. Pole je nepovinné.

- **Místo generování (InstrumentationScope)**

  Popisujeme, z jakého místa v kódu byl záznam/událost vygenerován. Záznamy/události pocházející z jednoho místa mají stejnou hodnotu. Pole je nepovinné.

- **Atributy (Attributes)**

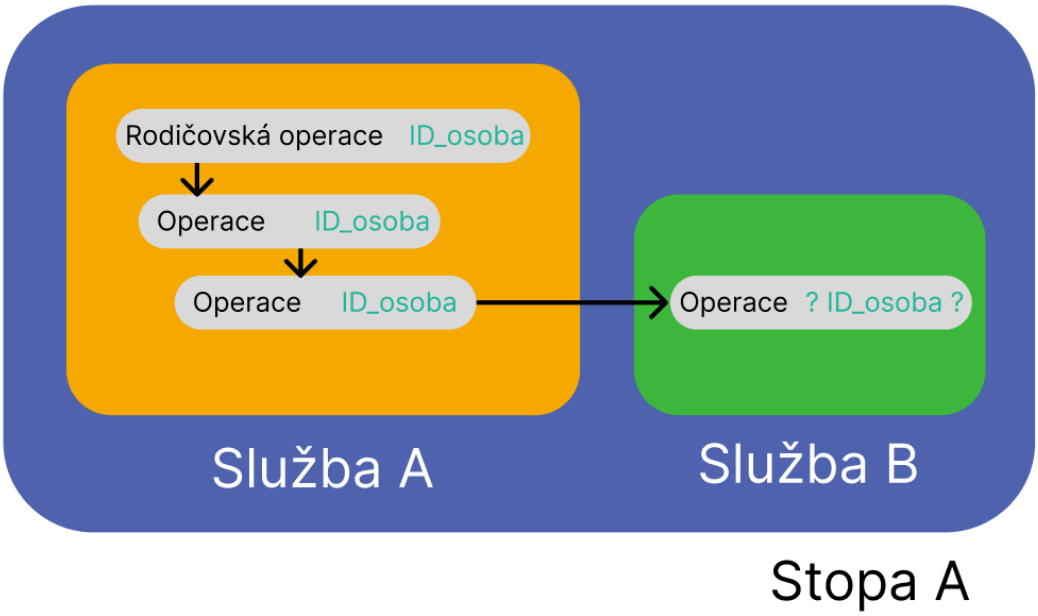
  Jedná se o dodatečné informace o události, jež nastala. Například se hodnota může lišit, i když událost nastala ve stejné aplikaci. V tom případě by nám nepomohlo pole `Zdroj`, ale mělo by nám pomoct pole `Atributy`. Můžeme zde uvádět informace například o verzi knihoven, které aplikace aktuálně využívá.

```
{
  "Timestamp": 1586960586000,
  "TraceId": "f4dbb3edd765f620",
  "SpanId": "43222c2d51a7abe3",
  "SeverityText": "INFO",
  "SeverityNumber": 9,
  "Body": "20200415T072306-0700 INFO I~like donuts",
  "Resource": {
    "service.name": "donut_shop",
    "service.version": "semver:2.0.0",
    "k8s.pod.uid": "1138528c-c36e-11e9-a1a7-42010a800198",
  },
  "Attributes": {
    "http.status_code": 500,
    "http.url": "http://example.com",
    "my.custom.application.tag": "hello",
  }
}
```

Výpis 4: Ukázka záznamu v JSON formátu

2.2.4 Sdílená data stopy (Baggage)

V systému OpenTelemetry signálem *Baggage* [8] označujeme informace, které se předávají mezi jednotlivými operacemi. Můžeme si je představit jako úložiště, jež obsahuje data ve formě klíč-hodnota. K těmto datům mají přístup pouze operace z jedné stopy. V systému OpenTelemetry pro předávání dat se používá již vysvětlená kontextová propagace. Například jako klíč použijeme text `id_osoby` a jako hodnotu unikátní klíč dané osoby `0108204256` v tomto případě rodné číslo.



Obrázek 2.3: Stopa s neznámým ID_osoby ve službě B

Na obrázku 2.3 vidíme jednu stopu pojmenovanou A. Představme si, že znázorňuje nákup lístků do kina, kde služba A představuje přidání lístku do košíku a služba B kontrolu počtu volných lístků. V jednotlivých operacích si zaznamenáváme, která osoba danou podoperaci provedla. V momentě kdy služba A zavolá službu B a chceme si poznamenat v operaci služby B Id osoby, která danou akci provedla, tak služba B nemá přístup k této informaci. V tom případě musíme využít sdílená data. Na začátku si tam uložíme Id osoby a poté ho můžeme kdykoliv ve stopě A přečíst a zapsat jej, kde potřebujeme.

Proč používat sdílená data

Jelikož systém OpenTelemetry je nezávislý na platformě a použitém programovacím jazyku, tak díky sdílení dat docílíme toho, že můžeme analyzovat, číst a používat data ve všech aplikacích bez ohledu na to, ve kterém programovacím jazyce byly napsány.

To je důležité, když vytváříme velký monitorovací systém, kde jsou aplikace napsány v různých programovacích jazycích. Tím poskytujeme vývojářům aplikací volnou ruku při výběru technologie pro vývoj.

Přidáváním dodatečných dat do signálů v navazujících službách, nám zajistí lepší filtrování a vyhledávání v monitorovacích nástrojích.

Vhodné použití

Ve sdílených datech by se měla uchovávat jen necitlivá data, protože mohou být vystavena třetím stranám. Vhodná data jsou například:

- identifikátor osoby
- IP adresa odesílatele
- verze webového serveru

2.3 Sběr telemetrických dat

Systém OpenTelemetry nám umožňuje sbírat telemetrická data dvěma způsoby. Podporuje manuální a automatický sběr nebo můžeme kombinovat oba způsoby tím, že použijeme automatický sběr a manuálně přidáváme informace do existujících dat nebo vytváříme úplně nová. V následujících dvou podkapitolách si popíšeme jednotlivé způsoby podrobněji a ukážeme si názornou ukázku v jazyce Python.

2.3.1 Automatický sběr dat

Využitím automatického sběru dat nám odpadá povinnost psát kód pro každou operaci v aplikaci. Stačí nám nainstalovat knihovnu `opentelemetry-distro`, která nám zařídí sběr telemetrických dat. Musíme také nastavit, kam budeme telemetrická data odesílat a to pomocí parametru při spouštění programu. Ukázku si můžeme prohlédnout v příloze A.

```
opentelemetry-instrument --traces_exporter console python app.py
```

Výpis 5: Příkaz pro spuštění programu app s výpisem stop do příkazové řádky

Nebo můžeme místo nastavit pomocí *environment variables*, kde nastavíme hodnoty `OTEL_TRACES_EXPORTER` a `OTEL_METRICS_EXPORTER`.

```
opentelemetry-instrument python app.py
```

Výpis 6: Příkaz pro spuštění programu app

2.3.2 Manuální sběr dat

Při manuálním sběru dat musíme přidat kód do aplikace pro jednotlivé operace, jež chceme pozorovat. Bohužel si ukážeme pouze stopy a metriky, protože záznamy jsou u programovacích jazyků ve vývoji nebo je zatím nepodporují vůbec. Ukázku si můžeme prohlédnout v příloze A.

V následujících ukázkách potřebujeme knihovny `opentelemetry-api` a `opentelemetry-sdk`, které budeme vyžívat k manuálnímu sběru dat.

Sběr stop

Na začátku musíme inicializovat `TracerProvider`, pomocí kterého vytváříme distribuované stopy a operace. Ve výpisu 7 vidíme také ještě výpis dat do příkazové řádky pomocí metody `ConsoleSpanExporter`.

```
from opentelemetry import trace
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import (
    BatchSpanProcessor,
    ConsoleSpanExporter,
)

provider = TracerProvider()
processor = BatchSpanProcessor(ConsoleSpanExporter())
provider.add_span_processor(processor)

# Nastavení globálního TraceProvideru
trace.set_tracer_provider(provider)

# Inicializace proměnné tracer pomocí, které budeme
# vytvářet jednotlivé operace
tracer = trace.get_tracer(__name__)
```

Výpis 7: Základní inicializace pro sběr stop

Ve výpisu 8 vidíme vytvoření distribuované stopy, která se skládá ze dvou operací. Jedné hlavní, která obsahuje vnořenou operaci.

```
def square():
    with tracer.start_as_current_span("parent") as parent:
        # rodičovská operace
        number = random()
        # Vytvoření vnořené operace
        with tracer.start_as_current_span("child") as child:
            return number * number
```

Výpis 8: Vytvoření distribuované stopy

Získání aktuální operace můžeme využít i při automatickém sběru dat a doplnit dodatečné informace k operaci. Získání aktuální operace, kterou budeme používat v dalších ukázkách pro přidávání dodatečných informací do operace vidíme ve výpisu 9.

```python
from opentelemetry import trace

current_span = trace.get_current_span()
```

Výpis 9: Získání aktuální operace

Ve výpisu 10 vidíme přidání vlastního atributu do operace nebo využití předdefinovaného jména atributu. Pro využití předem nadefinovaných jmen atributů musíme nainstalovat balíček `opentelemetry-semantic-conventions`.

```python
from opentelemetry.semconv.trace import SpanAttributes

# Vlastní atribut
current_span.set_attribute("some.value", 1)

# Předdefinovaný název atributu
current_span.set_attribute(SpanAttributes.HTTP_METHOD, "GET")
```

Výpis 10: Přidání atributu do operace

Ve výpisu 11 vidíme přidání události operace, která obsahuje zprávu čitelnou pro člověka.

```python
current_span.add_event("Some interesting message!")
```

Výpis 11: Přidání události do operace

Operace mohou mít žádné nebo více spojení. V tomto případě nemyslíme vnořené operace. Ve výpisu 12 vidíme kde pomocí kontextu jedné operace vytvoříme novou a tím definujeme spojení mezi nimi.

```python
contex = trace.get_current_span().get_span_context()

link = trace.Link(contex)

with tracer.start_as_current_span("new-span", links=[link]) as new_span:
    # Kód pro operaci, kterou zaznamenáváme
```

Výpis 12: Vytvoření spojení mezi dvěma operacemi

Pokud chceme ukládat v operaci informaci o návratovém kódu nebo chceme uložit zachycenou výjimku a její zprávu, využíváme k tomu statusu. Ve výpisu 13 vidíme uložení informace o výjimce a nastavení statusu.

```python
from opentelemetry.trace import Status, StatusCode

try:
    # Kód, který může selhat
except Exception as ex:
    current_span.set_status(Status(StatusCode.ERROR))
    current_span.record_exception(ex)
```

Výpis 13: Zaznamenání statutu a výjimky v operaci

Sběr metrik

Na začátku musíme inicializovat `MeterProvider`, pomocí kterého vytváříme metriky a nastavit kam budeme data odesílat. Ve výpisu 14 vidíme také ještě výpis dat do příkazové řádky pomocí metody `ConsoleMetricExporter`.

```python
from opentelemetry import metrics
from opentelemetry.sdk.metrics import MeterProvider
from opentelemetry.sdk.metrics.export import (
    ConsoleMetricExporter,
    PeriodicExportingMetricReader,
)

metric_reader = PeriodicExportingMetricReader(ConsoleMetricExporter())
provider = MeterProvider(metric_readers=[metric_reader])

# Nastavení globálního MeterProvideru
metrics.set_meter_provider(provider)

# Inicializace proměnné meter pomocí,
# které poté budeme vytvářet jednotlivé metriky
meter = metrics.get_meter(__name__)
```

Výpis 14: Základní inicializace pro sběr metrik

Nejprve si inicializujeme globální metriku, která se vytváří pouze jednou a poté voláme pouze metody podle dané metriky. Ve výpisu 15 si ukážeme počítadlo pro počet zavolání metody `square`.

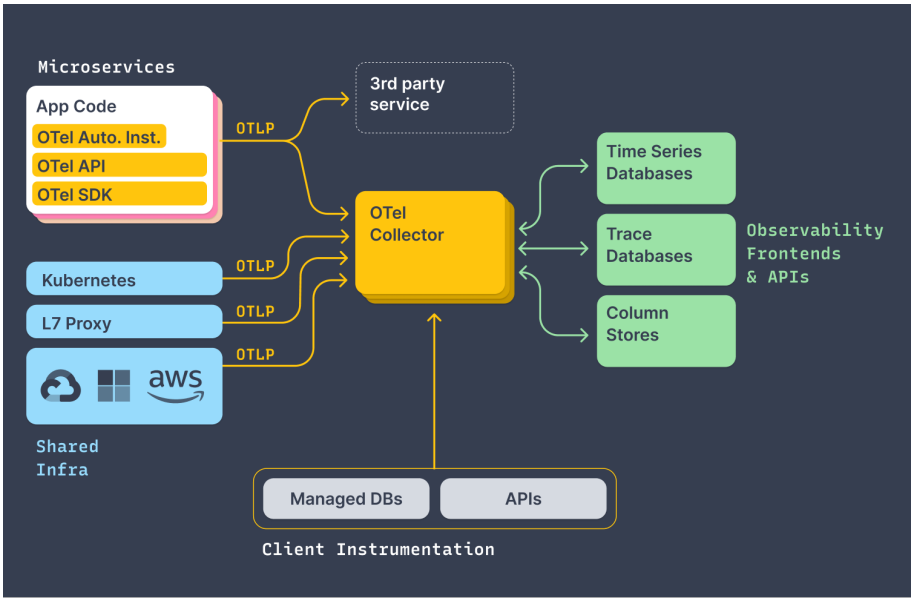
```
square_counter = meter.create_counter("square.counter", unit="1",
    description="Counts number calls of the method"
)

def square(number):
    # Zvýšíme počítadlo
    square_counter.add(1, {"square.number": number})
    return number*number
```

Výpis 15: Základní inicializace pro sběr metrik

2.4 Architektura

V systému OpenTelemetry se architektura skládá z několika hlavních komponentů, jež umožňují sběr, úpravu a odesílání telemetrických dat. Hlavní komponentou je kolektor (Collector), který přijímá data od všech poskytovatelů a odesílá je na monitorovací nástroje.



Obrázek 2.4: Doporučená architektura[8]

Na obrázku 2.4 vidíme architekturu, kde se uprostřed nachází kolektor (OTel Collector), který přijímá telemetrické signály pomocí OpenTelemetry protokolu (OTLP). Můžeme si povšimnout, že data nemusíme přijímat jen lokálně, kde máme nasazený kolektor, ale dokážeme data přijímat od poskytovatelů cloudových služeb, jako je například AWS[9], Azure[10] a mnoho dalších. Kolektor nadále data zpracuje a odesílá je do aplikace, kde jsou uživateli data zobrazeny a současně je ukládá do databáze.

[8] Obrázek byl převzat z webu https://opentelemetry.io/docs/ [29.11.2022]
[9] Amazon Web Services – poskytovatel cloudových služeb https://aws.amazon.com/ [29.11.2022]
[10] Microsoft Azure – cloudové výpočetní služby https://azure.microsoft.com/en-us/ [29.11.2022]

2.4.1 Kolektor (Collector)

Kolektor [1] nabízí způsob přijímaní, zpracování a odesílaní dat nezávisle na dodavateli. Tudíž odstraňuje potřebu mít vlastní sběrač dat pro každý protokol nebo dokonce pro jednotlivá telemetrická data. Podporuje příjem telemetrických dat od mnoha dodavatelů s otevřeným zdrojovým kódem, jako jsou například Jaeger, Fluent Bit, Prometheus a mnoho dalších. Telemetrická data po zpracování odesílá do jednoho nebo více monitorovacích nástrojů. Kolektor je výchozím místem, kam knihovny pro generovaní telemetrických dat odesílají svá data.

Obrázek 2.5: Ukázka architektury kolektoru[11]

Na obrázku 2.5 vidíme architekturu kolektoru, která se skládá ze tří částí. První je přijímač (Receivers), který dokáže přijímat data v různých formátech. V tomto případě přijímá dvoje telemetrická data pomocí OpenTelemetry protokolu (OTLP), která jsou naznačena zelenou a žlutou trasou. Druhou částí je procesor, který může manipulovat s odesílanými daty. Poslední částí je exportér (Exporters), který odesílá zpracovaná telemetrická data pomocí podporovaných protokolů do monitorovacích nástrojů. V tomto případě odesílá data zelené trasy pomocí Prometheus a OpenTelemetry protokolu a žluté trasy Jaeger protokolem. Nyní se podíváme na jednotlivé komponenty kolektoru podrobněji.

- **Přijímač (Receivers)**

  Přijímačem vstupují data do kolektoru. Existují dva druhy. První naslouchá a čeká na služby, až mu odešlou telemetrická data a druhý, který si stahuje data ze služeb. Jak již bylo řečeno, dokáže přijímat data v mnoha formátech. Ve výchozím stavu je přijímač nastaven na OpenTelemetry protokol. Po přijetí se data převedou do interního formátu kolektoru a jsou předána procesoru.

- **Procesor (Processors)**

[11] Obrázek byl převzat z webu https://raw.githubusercontent.com/open-telemetry/opentelemetry.io/main/iconography/Otel_Collector.svg [29.11.2022]

Procesor zpracovává nashromážděná telemetrická data před jejich předáním exportéru. Dokáže provést jakoukoliv úpravu dat při jejich průchodu. Pokud data mají být zveřejněna, můžeme odstranit osobní údaje uživatelů z telemetrických dat. Podporuje také dávkování dat před odesláním, jejich znovu odeslání, pokud exportér selže, přidávaní metadat a mnoho dalšího.

- **Exportér (Exporters)**

  Exportér umožňuje odesílání zpracovaných dat do monitorovacích nástrojů. Můžeme odesílat jedny telemetrická data do více nástrojů nebo je rozdělit. Například odesílat stopy do monitorovacího nástroje a záznamy vypisovat na standardní výstup.

Systém OpenTelemetry nám umožňuje odesílat data přímo do monitorovacích nástrojů. Tady nám vzniká otázka: „Proč bychom vlastně měli používat kolektor?“ Na tuto otázku si odpovíme následujícím výčtem výhod kolektoru.

- nezávislý sběr telemetrických dat na protokolu
- přebírá odpovědnost za správu telemetrických dat od aplikace
- umožňuje exportovat data v různých formátech do více monitorovacích nástrojů
- jednotná konfigurace sběru a odesílání telemetrických dat

2.4.2 Rozhraní (API)

Doposud jsme si popsali, jaká telemetrická data sbíráme, a jak vypadají, ale systém OpenTelemetry také definuje, jak s telemetrickými daty pracovat.

OpenTelemetry API specifikuje a popisuje třídy a jejich funkce pro interakci s telemetrickými daty. Například popisuje, jak systém OpenTelemetry může implementovat generování stop nebo hlásit metrická měření.

Specifikace API obsahuje dvě základní složky:

- definici rozhraní, které používáme pro práci s telemetrickými daty,
- minimální implementaci tohoto rozhraní.

2.4.3 Sada nástrojů pro generování telemetrických dat (SDK)

Systém OpenTelemetry nabízí pro podporované programovací jazyky sadu nástrojů, které umožňují používat OpenTelemetry rozhraní ke generování telemetrických dat a exportovat je do kolektoru, nebo přímo do monitorovacího nástroje.

Každá sada nástrojů by minimálně měla obsahovat:

- nezávislý sběr telemetrických dat na protokolu,
- přebírá odpovědnost za správu telemetrických dat od aplikace,
- umožňuje exportovat data v různých formátech do více monitorovacích nástrojů.

2.5 Protokol OTLP

Systém OpenTelemetry definuje specifikaci protokolu OTLP (OpenTelemetry protokol) [11] pro přenos telemetrických dat bez ohledu na monitorovací nástroj a poskytovatele dat. Díky této vlastnosti je výměna monitorovacího nástroje snadná.

Pomocí OTLP můžeme přenášet telemetrická data z SDK do kolektoru a z kolektoru do monitorovacího nástroje. Specifikace OTLP definuje kódování, transport a mechanismus doručování telemetrických dat.

2.5.1 Detaily protokolu

OpenTelemetry protokol definuje kódování telemetrických dat a protokol používaný k výměně dat mezi klientem a serverem. Tato specifikace definuje jak OpenTelemetry protokol implementuje transport dat přes gRPC[12] a HTTP 1.1. Specifikuje také schéma pro serializaci dat pomocí Protocol Buffers[13].

Komunikuje pomocí požadavku a odpovědi, kde klient pošle požadavek na server a dostane ze serveru odpověď.

Všechny servery, které chtějí používat OpenTelemetry protokol, musí podporovat tyto metody komprese:

- bez komprese,
- Gzip komprese.

2.5.2 Protokoly OTLP a gRPC

Po navázání úvodního spojení gRPC, začne klient odesílat telemetrická data unárními požadavky[14] pomocí tzv. *ExportServiceRequest* zpráv.

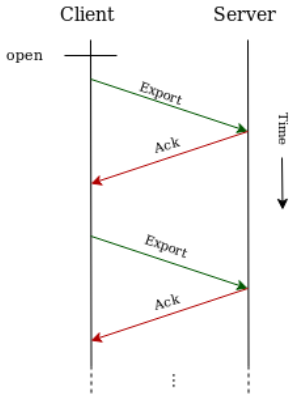
- Pro záznamy se nazývá *ExportLogsServiceRequest*
- Pro stopy se nazývá *ExportTraceServiceRequest*
- Pro metriky se nazývá *ExportMetricsServiceRequest*

Komunikace vždy probíhá jednou dvojicí klientem a serverem. Klient průběžně odesílá požadavky na server a čeká na potvrzení o přijetí. Výchozí port je 4317.

[12] Vzdálené volání procedur `https://grpc.io/` [4.12.2022]

[13] Mechanizmus pro serializaci strukturovaných dat `https://developers.google.com/protocol-buffers/docs/overview` [4.12.2022]

[14] Unární komunikace je tradiční komunikace požadavek-odpověď používaný také v protokolu HTTP.



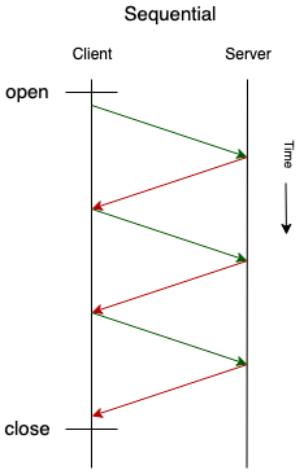
Obrázek 2.6: Ukázka unární komunikace mezi klientem a serverem[15]

Během požadavku mezi klientem a serverem, by nemělo dojít ke ztrátě dat. Záruka není garantována, pokud data putují přes více uzlů, jako například: aplikace > agent > kolektor > monitorovací nástroj.

Sekvenční a souběžné požadavky

Protokoly OTLP a gRPC podporují souběžné a sekvenční odesílání požadavků.

Sekvenční odesílaní znamená, kdy klient odešle jeden požadavek a čeká na potvrzení od serveru, než odešle další. Komunikace takhle pokračuje, než se spojení ukončí. Tento způsob komunikace se doporučuje, když vyžadujeme jednoduchou implementaci a server a klient se nachází v síti s malou latencí.

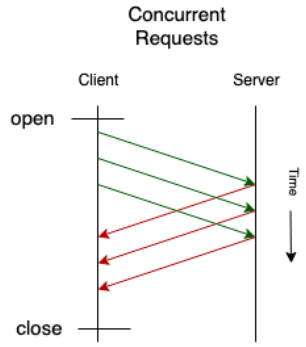


Obrázek 2.7: Ukázka sekvenční komunikace[16]

Souběžné odesílání používáme, když chceme dosáhnout velké propustnosti. Klient tedy nemusí čekat před odesláním nového požadavku, než mu přijde potvrzení na předchozí.

[15]Obrázek byl převzat z webu https://opentelemetry.io/docs/reference/specification/protocol/img/otlp-request-response.png [19.12.2022]

[16]Obrázek byl převzat z webu https://opentelemetry.io/docs/reference/specification/protocol/img/otlp-sequential.png [19.12.2022]



Obrázek 2.8: Ukázka souběžných komunikace[17]

Spojení ukončujeme tím, že klient počká na všechna potvrzení od serveru nebo až vyprší časový limit, jež je specifický pro implementaci. Tím je zajištěno spolehlivé doručení telemetrických dat. Pokud klient není schopen doručit požadavek serveru, například při čekaní na potvrzení od serveru a vyprší u toho časový limit. Klient by měl zaznamenat danou skutečnost, že data nebyla doručena.

Odpověď ze serveru

Odpověď je reakce serveru na klientův požadavek. Odpověď musí mít určitý formát, který si popíšeme v následujících bodech.

- **Úspěch (Full Success)**

  Úspěšnou zprávu můžeme očekávat, pokud byla všechna telemetrická data přenesena nebo byl požadavek prázdný. V obou případech odesíláme úspěšnou odpověď.

- **Částečný úspěch (Partial Success)**

  Částečný úspěch nastane, když server přijme jenom část telemetrických dat a zbytek odmítne. Ve zprávě inicializuje pole *partial_success* a nastavíme *rejected_spans*, *rejected_data_points* a *rejected_log_records* podle počtu odmítnutých stop, metrik a záznamů. Server podle chyby může vyplnit pole *error_message* se zprávou čitelnou pro člověka, která popisuje chybu, jak ji může vyřešit a kde najít další informace.

- **Selhání (Failure)**

  Selhání dělíme na dvě kategorie: opravitelné a neopravitelné.

  - Opravitelné selhání znamená, když server není schopen zpracovat telemetrická data. Klient obdrží zprávu se selháním, ale s informací, že může opakovat odeslaní dat. Server označí zprávu pomocí kódu *Unavailable* a můžeme poskytnout další informace pomocí statusu *RetryInfo*.
  - Neopravitelné selhání znamená, když nastane chyba například při deserializaci špatných dat. Klient obdrží zprávu o neopravitelném selhání, takže telemetrická data nesmí být odeslána znova. Klient musí daná data zahodit a zvýšit počítadlo,

[17]Obrázek byl převzat z webu https://opentelemetry.io/docs/reference/specification/protocol/img/otlp-concurrent.png [19.12.2022]

kde si uchovává kolik telemetrických dat zahodil. Server musí zprávu označit, aby klient poznal, že se jedná o neopravitelnou chybu. Označí ji pomocí *InvalidArgument* a další informace můžeme poskytnout pomocí statusu *BadRequest*.

Zpomalení požadavků (Throttling)

Protokol OpenTelemetry podporuje signalizaci přetížení. Pokud server není schopen přijímat tolik dat, musí danou situaci oznámit klientovi, aby se zabránilo přetížení serveru. Server oznámí tuto událost pomocí chyby s kódem *Unavailable*.

2.5.3 Protokoly OTLP a HTTP

Protokol používá Protobuf kódování pro přenos telemetrických dat v binární formě nebo ve formátu JSON. Výchozí port je 4318.

Pro přenos telemetrických dat využívá POST požadavků z klienta na server. Můžeme využívat protokol HTTP/1.1 nebo HTTP/2, ale vždy daná implementace musí podporovat HTTP/1.1, kdyby došlo k selhání HTTP/2.

Kódování binárních dat pomocí Protobuf

Telemetrická data v binární podobě jsou zakódovaná pomocí standardu proto3[18]. Aby klient a server věděl, že dané kódování využíváme, musíme mít nastaveno v HTTP hlavičce `Content-Type: application/x-protobuf` u požadavku i odpovědi.

Kódování dat ve formátu JSON pomocí Protobuf

Využívá se stejný standard jako u dat v binární podobě a využívá se tzv. JSON mapovaní mezi Protobuf formátem a JSON formátem, ale pro potřeby OpenTelemetry protokolu je zde pár odchylek:

- hodnoty `trace_id` a `span_id` jsou reprezentovány pomocí řetězce bez rozlišení velkých a malých písmen v base16[19],
- výčtové typy (enum) mohou být převedeny pouze jako celé číslo,
- klíče v JSON objektu jsou převáděny na tzv. camel case.

Stejně jako u binárních dat musí být u klienta i serveru nastaven u požadavků a odpovědí `Content-Type: application/x-protobuf`.

HTTP požadavky

Telemetrická data posíláme pomocí HTTP POST požadavků. Jsou nadefinované výchozí cesty pro jednotlivá telemetrická data. Například adresa serveru je `telemetrydata.com`. Cesta pro odeslání stop by byla `https://telemetrydata.com/v1/traces`.

Výchozí cesty pro jednotlivá telemetrická data:

[18] Specifikace standardu https://developers.google.com/protocol-buffers/docs/proto3
[19] Specifikace base16 https://www.rfc-editor.org/rfc/rfc4648#section-8 [21.12.2022]

- pro stopy `/v1/traces`, kde tělo požadavku je tvořeno pomocí *ExportMetricsServiceRequest*,
- pro stopy `/v1/logs`, kde tělo požadavku je tvořeno pomocí *ExportLogsServiceRequest*,
- pro stopy `/v1/metrics`, kde tělo požadavku je tvořeno pomocí *ExportMetricsServiceRequest*.

Klient může tělo požadavku zabalit pomocí gzip komprese, ale v tom případě musí přidat další hlavičku do HTTP požadavku `Content-Encoding: gzip`.

Pokud budeme chtít dosáhnout větší propustnosti, můžeme vytvořit více paralelních připojení HTTP. Počet připojení by měl být nastavitelný, aby nedošlo k přetížení.

HTTP odpověď

Tělo odpovědi musí být serializovatelné pomocí protokolu Protobuf. Hlavičky odpovědi nastavuje podle typu odpovědi, jestli tělo obsahuje binární data, tak se nastaví hlavička `Content-Type: application/x-protobuf` a pokud server odesílá odpověď ve formátu JSON tak `Content-Type: application/json`.

Pokud server obdrží požadavek s hlavičkou `Accept-Encoding: gzip`, může odpověď použít kompresi gzip na tělo odpovědi. Odpověď může mít jeden z následujících tvarů.

- **Úspěch (Full Success)**

  Úspěšnou zprávu můžeme očekávat, pokud byla všechna telemetrická data přenesena, nebo byl požadavek prázdný. V obou případech odesíláme úspěšnou odpověď pomocí HTTP 200 OK odpovědi.

- **Částečný úspěch (Partial Success)**

  Částečný úspěch nastane, když server přijme jenom část telemetrických dat a zbytek odmítne. Musíme odeslat odpověď pomocí HTTP 200 OK. Ve zprávě musíme inicializovat pole *partial_success* a nastavit *rejected_spans*, *rejected_data_points* a *rejected_log_records* podle počtu odmítnutých stop, metrik a záznamů. Server podle chyby může vyplnit pole *error_message* se zprávou čitelnou pro člověka, která popisuje chybu, jak ji může vyřešit a kde najít další informace.

- **Selhání (Failure)**

  Selhání zpracování požadavku musíme nahlásit klientovi pomocí příslušných HTTP status kódů 4xx nebo 5xx. Tělo odpovědi musí obsahovat zprávu, jež popisuje problém.

- **Špatná data**

  Pokud se při zpracování požadavku zjistí, že obsahuje špatná data. Server musí danou událost ohlásit pomocí odpovědi se status kódem HTTP 400 Bad Request. Pole *Status.details* musí obsahovat zprávu s Bad Requestem a popis, která data jsou špatná. Pokud klient obdrží odpověď se statusem HTTP 400, nesmí nikdy požadavek znovu odeslat.

Zpomalení požadavků (Throttling)

Pokud doje k přetížení serveru, musí server danou situaci oznámit klientovi pomocí protokolu HTTP se statusem 429 Too Many Requests nebo HTTP 503 Service Unavailable. Odpověď může obsahovat hlavičku `Retry-After`, jež nám doporučuje časový interval v sekundách, kdy má dojít k zopakování odeslání požadavku. Pokud odpověď danou hlavičku neobsahuje. Poté by měl klient odesílání požadavku opakovat s intervalem, co si určí, který bude exponenciálně růst.

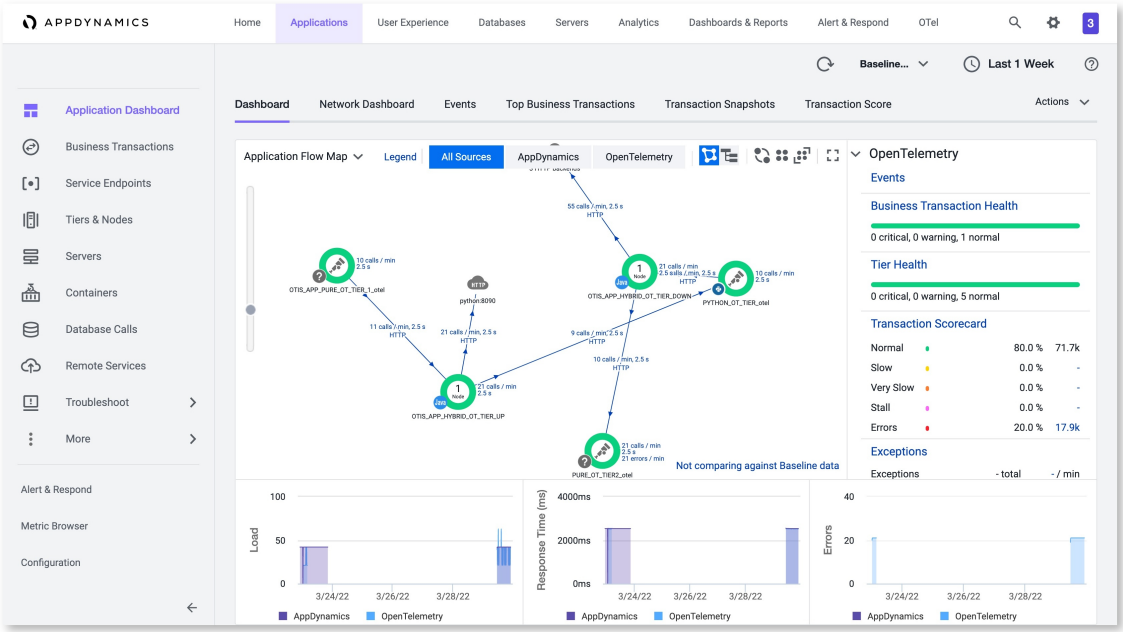
2.6 Podpora OpenTelemetry od třetích stran

Systém OpenTelemetry nám přináší jednotný způsob jak sbírat telemetrická data z aplikací napsané ve většině populárních programovacích jazycích. Proto se rozhodlo mnoho firem přidat podporu systému OpenTelemetry do jejich monitorovacích nástrojů. V dalších podkapitolách si popíšeme pár populárních nástrojů podporující OpenTelemetry.

2.6.1 Firma AppDynamics

Dceřiná firma Cisco Systems, která vyvíjí monitorovací nástroj, který primárně umožňuje přijímat stopy a zobrazovat je uživateli pomocí webového rozhraní. Pro systém OpenTelemetry se služba nazývá *AppDynamics for OpenTelemetry*. Pro zobrazování záznamů a metrik je potřeba nainstalovat rozšíření pro jejich podporu. Jedná se o placenou službu.

Odesílání telemetrických dat můžeme docílit pomocí jednoduché konfigurace kolektoru a OpenTelemetry protokolu. V kolektoru pomocí procesoru přidáme telemetrickým datům tyto tři atributy: *appdynamics.controller.account*, *appdynamics.controller.host* a *appdynamics.controller.port*. Exportéru nastavíme adresu, na kterou data mají být odesílána, a přidáme hlavičku s klíčem, který si vygenerujeme ve webovém rozhraní.



Obrázek 2.9: Ukázka webového rozhraní AppDynamics[20]

2.6.2 Poskytovatel cloudových služeb Amazon Web Services (AWS)

Firma AWS si vytvořila vlastní distribuci OpenTelemetry s podporou původních tvůrců. Rozšířila systém o sběr metadat z cloudových zdrojů AWS a jejich služeb. Můžeme tedy korelovat data o výkonu aplikace a o infrastruktuře, na které běží, tím pádem zkrátíme průměrnou dobu do vyřešení problému, pokud nastane. Pomocí AWS distribuce můžeme sbírat telemetrická data, když aplikace běží v následujících prostředí.

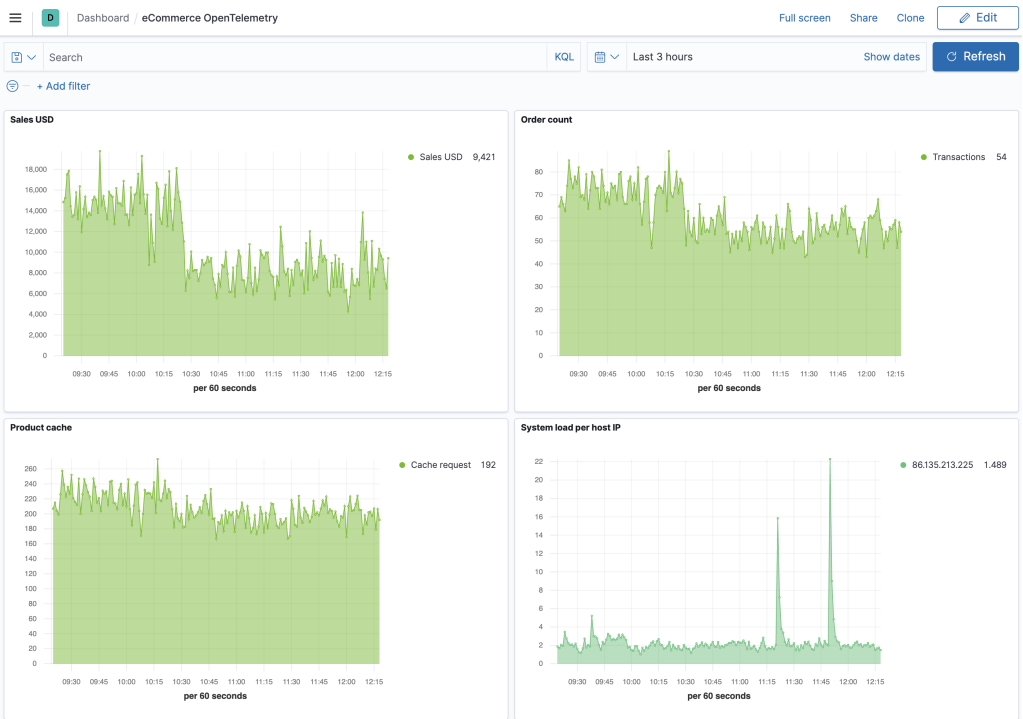
- Amazon Elastic Compute Cloud (EC2)
- Amazon Elastic Container Service (ECS) a Amazon Elastic Kubernetes Service (EKS)
- AWS Fargate
- AWS Lambda

Je zde ale jedna nevýhoda, musíme využít monitorovacích nástrojů od AWS (AWS X-Ray, Amazon CloudWatch, Amazon Managed Service for Prometheus) nebo jejich partnerů.

2.6.3 Elastic

Elastic je monitorovací nástroj, který je úplně zdarma. Umožní nám rychle vizualizovat všechna telemetrická data ve webovém rozhraní, data ze systému OpenTelemetry pomocí Elastic Stacku.

Jelikož podporuje nativně OpenTelemetry protokol, stačí nám pouze v exportéru nastavit koncový bod monitorovacího nástroje a autorizaci pomocí vygenerovaného Api klíče.



Obrázek 2.10: Ukázka webového rozhraní Elastic zobrazující metriky[21]

[20]Obrázek byl převzat z webu https://cdn.brandfolder.io/5OKQXSAT/at/hfm5nm7rqbrn6298rbrtqbf/Application_Dashboard.jpeg [25.12.2022]

2.6.4 Projekt Grafana Labs

Monitorovací nástroj s otevřeným zdrojovým kódem. Podporuje vizualizaci všech telemetrických dat, které nabízí systém OpenTelemetry. Zobrazit si vizualizace můžeme ve webovém rozhraní. Nabízí zdarma doživotní přístup pro jednoho uživatele při maximální zátěži 10 tisíc aktuálně zpracovávaných požadavků a 50GB dat pro záznamy.

Telemetrická data můžeme odesílat pomocí protokolu OpenTelemetry a to třemi způsoby.

- Pomocí kolektoru, kde v rozšíření nastavíme autentizaci a v exportéru na ní odkážeme.
- Odesílat data přímo z aplikace do Grafana Cloud. Doporučuje se pouze pro testování.
- Pomocí Grafana agenta, který poskytuje koncový bod OTLP pro příjem dat. Zatím je tato funkce v raném vývoji, a proto se nedoporučuje použít v produkci.

2.6.5 Firma Splunk

Softwarová firma Splunk vyvíjí monitorovací nástroje podporující telemetrická data ze systému OpenTelemetry. Splunk APM je monitorovací nástroj pro vizualizaci dat z aplikací a Splunk Infrastructure Monitoring je monitorovací nástroj pro vizualizaci dat o infrastruktuře na které mohou aplikace běžet.

Splunk také vyvinul vlastní knihovny pro automatický sběr telemetrických dat z aplikace pro programovací jazyky Java a Python.

Pro odesílání dat na monitorovací nástroj máme předpřipravený kolektor od firmy Splunk, jež podporuje protokoly pro příjem dat, jako jsou OpenTelemetry protokol, Jaeger, Zipkin a Prometheus.

[21] Obrázek byl převzat z webu https://www.elastic.co/guide/en/apm/guide/current/legacy/guide/images/ecommerce-dashboard.png [26.12.2022]

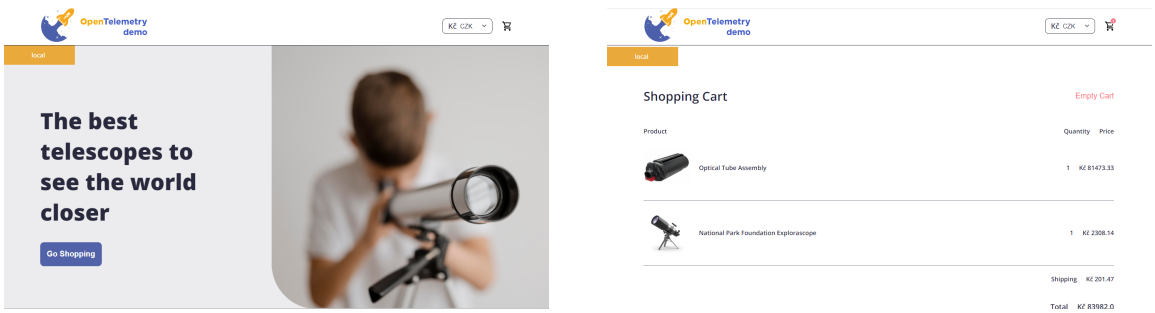
Kapitola 3

Testovací prostředí

Systém OpenTelemetry podporuje velké množství programovacích jazyků a proto jsem se rozhodl použít demo aplikaci přímo od tvůrců systému OpenTelemetry. Aplikace je navržena pomocí mikroslužeb[1]. Díky této architektuře využijeme většinu programovacích jazyků, podporovaných systémem OpenTelemetry.

3.1 Demo aplikace

Aplikace představuje on-line obchod s hvězdářskými potřebami, kde uživatel může přidat produkt do košíku, vybrat měnu, ve které bude nakupovat, a košík s produkty objednat.



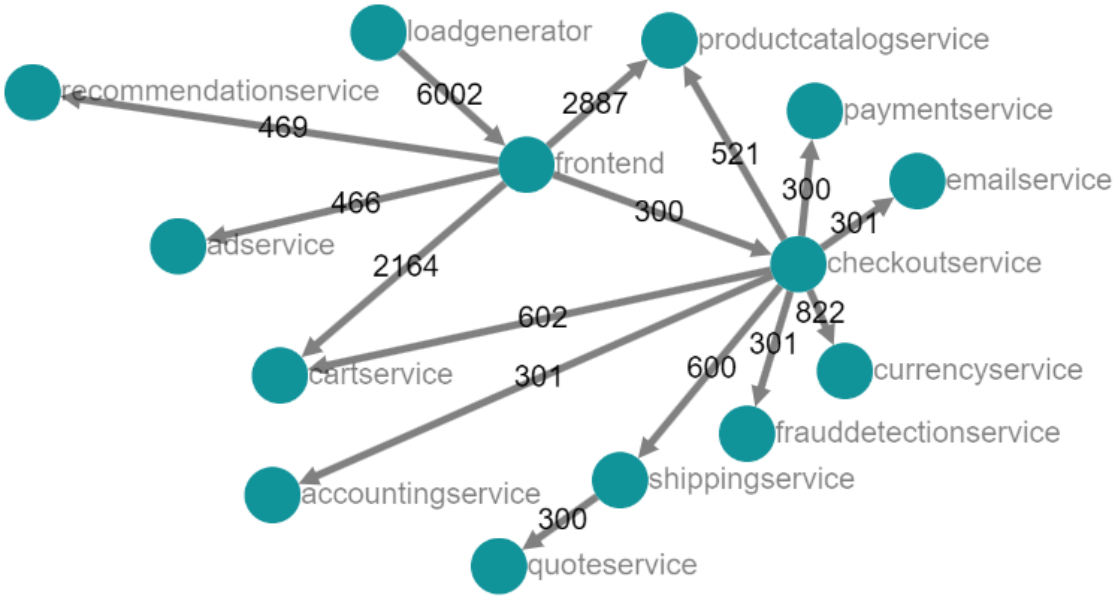
Obrázek 3.1: Ukázka obchodu

3.1.1 Architektura aplikace

Aplikace je založena na architektuře mikroslužeb. Skládá se z 14 služeb, kde každá obstarává určitou funkcionalitu webové aplikace a je napsána jinou technologií, aby se pokryla větší část knihoven pro generování telemetrických dat systému OpenTelemetry.

Nachází se zde ještě jedna služba, která má simulovat uživatele a jejich pohyb po webové aplikaci.

[1] Jedná se o architekturu, kde výsledná aplikace je rozdělena do několika menších. Přináší to výhodu, že na každé části může pracovat jiný tým, který si může zvolit vlastní nástroje k vývoji.



Obrázek 3.2: Ukázka architektury aplikace a komunikace mezi jednotlivými službami

Na obrázku 3.2 vidíme jednotlivé služby, komunikaci mezi nimi a počet požadavků. Směr šipek nám udává, která služba odeslala požadavek. Můžeme si všimnout, že danou architekturu nám vykreslil monitorovací nástroj Jaeger. Dále si popíšeme hlavní služby podrobněji.

- **Ad Service**

  Jedná se o službu napsanou v jazyce Java s použitím automatického generování telemetrických dat. Tato služba určuje vhodné reklamy, jež se mají uživateli zobrazovat. Používáme knihovnu i pro manuální generování telemetrických dat pro doplnění například atributů do operací.

- **Cart Service**

  Jedná se o službu napsanou v jazyce .NET s použitím automatického generování telemetrických dat. Tato služba uchovává položky vložené do košíku.

- **Checkout Service**

  Jedná se o službu napsanou v jazyce Go s použitím automatického generování telemetrických dat. Tato služba je zodpovědná za zpracování objednávek od uživatele.

- **Currency Service**

  Jedná se o službu napsanou v jazyce C++. Tato služba se stará o převod jedné měny na jinou.

- **Email Service**

  Jedná se o službu napsanou v jazyce Ruby s použitím automatického generování telemetrických dat. Tato služba odesílá email s potvrzením objednávky.

- **Fraud Detection Service**

  Jedná se o službu napsanou v jazyce Kotlin s použitím automatického generování telemetrických dat. Tato služba má na starost detekci podvodných objednávek.

- **Frontend**

  Jedná se o službu napsanou v jazyce JavaScript s použitím automatického generování telemetrických dat. Tato služba je zodpovědná za poskytnutí uživatelského rozhraní a komunikaci s API.

- **Payment Service**

  Jedná se o službu napsanou v jazyce JavaScript s použitím automatického generování telemetrických dat. Tato služba je zodpovědná za zpracování plateb platební kartou a kontrolu validity platební karty.

- **Product Catalog Service**

  Jedná se o službu napsanou v jazyce Go s použitím automatického generování telemetrických dat. Tato služba zodpovídá za práci s produkty v obchodě.

- **Quote Service**

  Jedná se o službu napsanou v jazyce PHP. Tato služba počítá poštovné na základě položek obsažené v košíku.

- **Recommendation Service**

  Jedná se o službu napsanou v jazyce Python s použitím automatického generování telemetrických dat. Tato služba doporučuje produkty pro uživatele na základě jeho aktivity.

- **Shipping Service**

  Jedná se o službu napsanou v jazyce Rust. Tato služba poskytuje informace o ceně a informace o přepravě objednávky.

3.1.2 Simulace uživatelů

Simulaci chování uživatele na webové aplikaci zajišťuje služba *Load Generator*. Tato služba je napsaná v jazyce Python a pro generování požadavků využívá knihovnu *Locust*[2]

[2]Knihovna Locust https://docs.locust.io/en/stable/writing-a-locustfile.html [30.12.2022]

Type	Name	# Requests	# Fails	Median (ms)	90%ile (ms)	99%ile (ms)	Average (ms)	Min (ms)	Max (ms)	Average size (bytes)	Current RPS
GET	/	26	3	57	160	12000	948	5	12064	39732	0.1
GET	/api/cart	87	2	35	50	91	34	5	91	23	0.2
POST	/api/cart	159	0	20	49	570	36	8	576	116	0.2
POST	/api/checkout	56	0	95	317000	360000	56351	65	360135	1611	0
GET	/api/data/?contextKeys=accessories	11	0	42	76	12000	1163	13	12383	303	0.1
GET	/api/data/?contextKeys=assembly	7	0	49	110	110	50	12	109	89	0.1
GET	/api/data/?contextKeys=binoculars	14	0	57	73	81	56	14	81	83	0

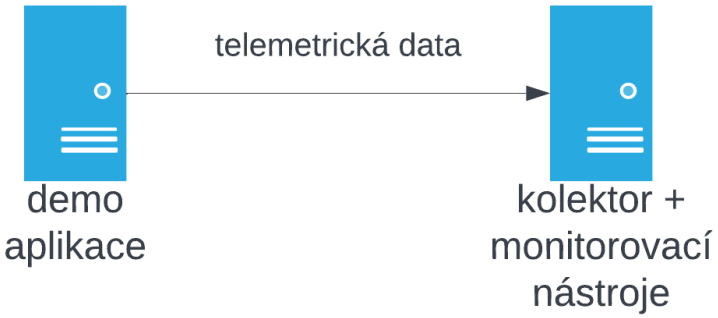
Obrázek 3.3: Ukázka webového rozhraní pro simulaci uživatelů

Na obrázku 3.3 vidíme požadavky, které generujeme a jejich statistiky. V horní liště můžeme měnit počet simulovaných uživatelů. Nyní vidíme, že na webu simulujeme chování 10 uživatelů. Vedle se nachází hodnota, jež udává počet požadavků za sekundu a procento chybných požadavků. Můžeme zde generování požadavků přerušit nebo vymazat statistiky.

3.2 Testovací prostředí

Pro testování jsem oddělil aplikaci a kolektor s monitorovacími nástroji na dva různé servery. Umožní nám to odchytávat komunikaci a zkoumat jednotlivé pakety, zahlcení sítě komunikací při odesílání dat kolektoru a mnoho dalších způsobů testování daného systému.

Architekturu testovacího prostředí můžeme lehce změnit díky využití kontejnerizace. Každá služba má vlastní kontejner a tak není problém nasadit danou část na odlišný server.



Obrázek 3.4: Ukázka testovacího prostředí

Zachytával jsem pakety v různých časových intervalech a při různém počtu požadavků, které jsem ovlivňoval pomocí změny počtu simulovaných uživatelů.

Počet uživatelů	Doba v minutách		
	10	**30**	**60**
1	11 250	28 765	57 528
10	21 310	64 246	128 031
50	44 599	136 560	245 422

Tabulka 3.1: Počet zachycených paketů

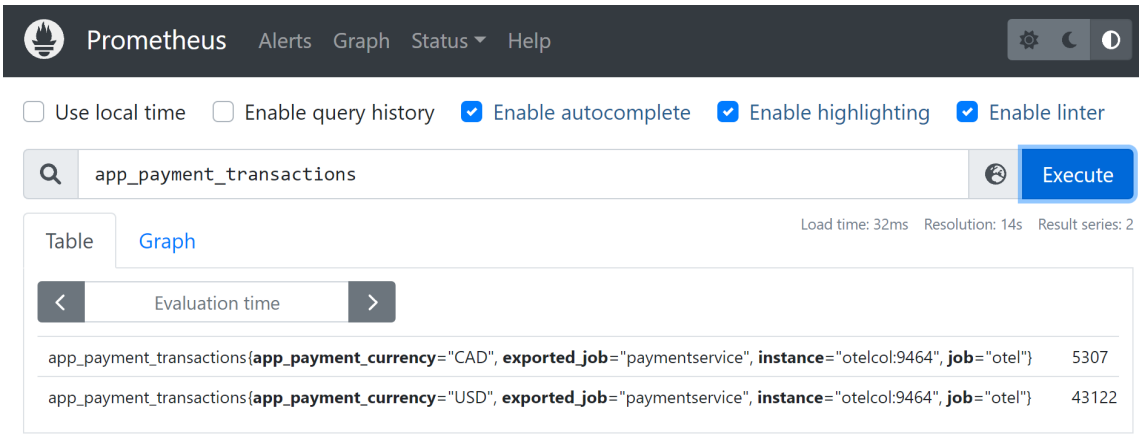
V tabulce 3.1 vidíme celkový počet zachycených paketů. Pro sběr komunikace jsem měl nastavený filtr na porty 4317 a 4318. V počtech paketů jsou započítána odesílaná telemetrická data a potvrzení od serveru o jejich přijetí.

3.3 Monitorovací nástroje

Telemetrická data vizualizujeme pomocí tří monitorovacích nástrojů. Stopy odesíláme do monitorovacího nástroje *Jaeger* a metriky do nástroje *Prometheus*. Poté všechna telemetrická data využívá nástroj *Grafana*.

3.3.1 Prometheus

Monitorovací nástroj Prometheus nám zobrazuje metriky. Vidíme všechny přijaté metriky a můžeme si je vizualizovat do panelů podle našich preferencí. Můžeme si je zobrazit jako hodnoty nebo v grafu, jak se vyvíjely v čase.

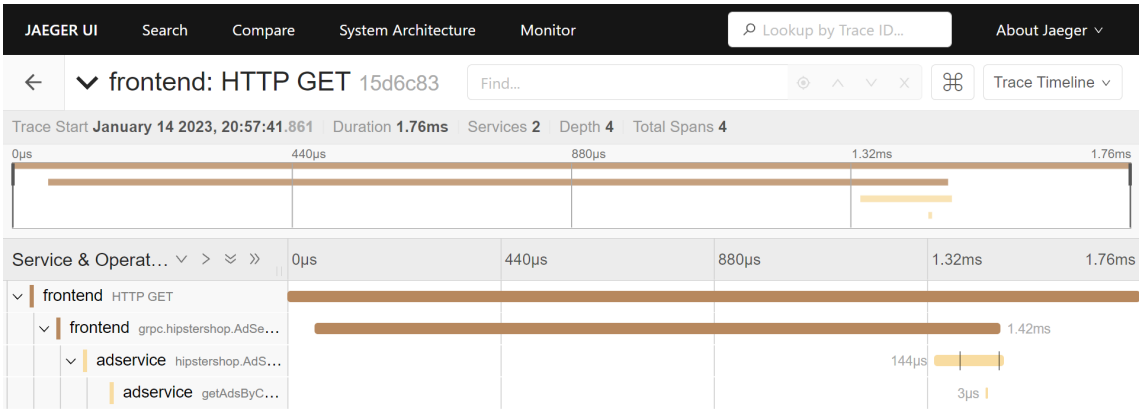


Obrázek 3.5: Ukázka monitorovacího nástroje Prometheus

Na obrázku 3.5 vidíme metriku, kde je vypsaná celková částka, která byla utracena za objednávky ve stejné měně.

3.3.2 Jaeger

Monitorovací nástroj Jeager nám umožňuje vizualizovat distribuované stopy. Nabízí výpis stop pro jednotlivé služby, porovnání dvou stop, ukázku architektury aplikace nebo monitorování služeb.

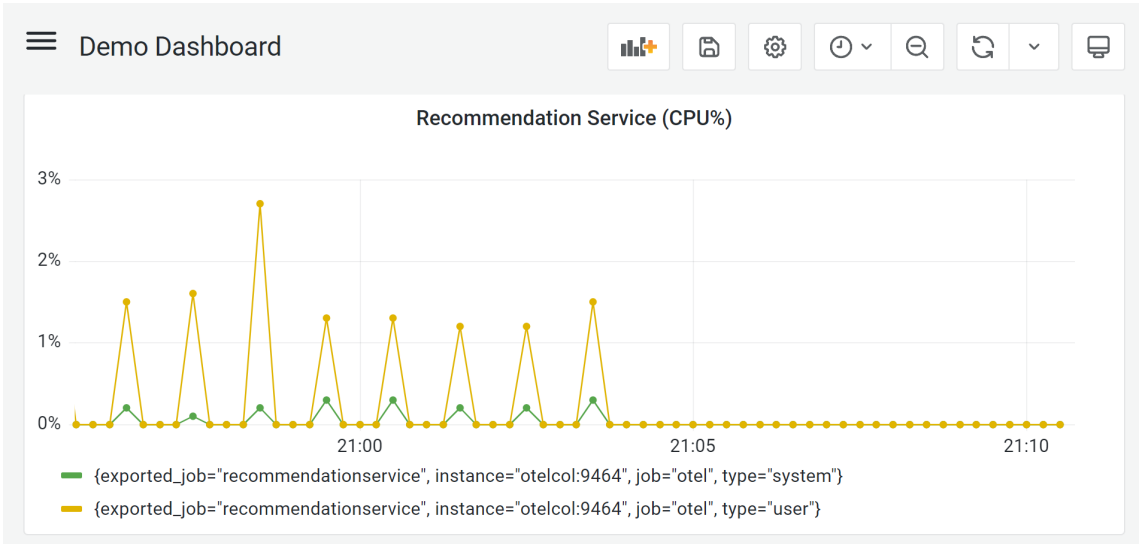


Obrázek 3.6: Ukázka monitorovacího nástroje Jaeger

Na obrázku 3.6 vidíme distribuovanou stopu se čtyřmi operacemi a informacemi, které nám mohou být užitečné při hledání problému nebo optimalizaci programu, jako je například doba trvání jednotlivých operací.

3.3.3 Grafana

Monitorovací nástroj Grafana nám umožňuje vizualizovat všechna telemetrická data, jež podporuje systém OpenTelemetry. Uživateli, který bude sledovat telemetrická data, umožňuje vytvořit nástěnky s panely podle jeho preferencí.



Obrázek 3.7: Ukázka monitorovacího nástroje Grafana

Na obrázku 3.7 vidíme graf, který znázorňuje využití procesoru službou *Recommendation Service*.

Kapitola 4

Experimenty s testovacím prostředím

V této kapitole si ukážeme jak zachytit a přečíst pakety systému OpenTelemetry. Provedeme experimenty v testovacím prostředí a podíváme se, jak telemetrická data vypadají. Cílem této kapitoly je získat představu, jaké události můžeme detekovat pomocí dat ze systému OpenTelemetry a jak data vypadají na síti.

4.1 Zachycení komunikace

Pro zachycení komunikace využíváme nástroj `tcpdump`[1], který slouží pro sběr a analýzu síťové komunikace v reálném čase a je dostupný na většině Unixových operačních systémech.

```
tcpdump -i enp2s0 'port 4317 or 4318' -w packetsOTLP.pcap
```

Výpis 16: Příkaz pro zachycení komunikace

Ve výpisu 16 vidíme zachycení komunikace na rozhraní `enp2s0`. Filtrujeme komunikaci pomocí portů 4317 a 4318, na kterých se nachází komunikace systému OpenTelemetry, a ukládáme zachycenou komunikaci do souboru `packetsOTLP.pcap`.

4.2 Čtení dat z komunikace

Zachycenou komunikaci přečteme za využití nástroje `Wireshark`[2]. Používá se ke sběru a analýze paketů, ale na rozdíl od nástroje `tcpdump` nabízí uživateli přívětivé grafické rozhraní.

Systém OpenTelemetry vyžívá protokoly gRPC a HTTP2. Pro serializaci dat využívá Protocol Buffers ve výchozím nastavení. Wireshark v základním nastavení nedokáže zobrazit zachycená data v čitelné podobě pro člověka. Musíme přidat `.proto` soubory, pomocí kterého Wireshark deserializuje data do čitelné podoby pro člověka.

[1] Oficiální stránky nástroje tcpdump https://www.tcpdump.org/ [28.03.2023]
[2] Oficiální stránky nástroje Wireshark https://www.wireshark.org/ [28.03.2023]

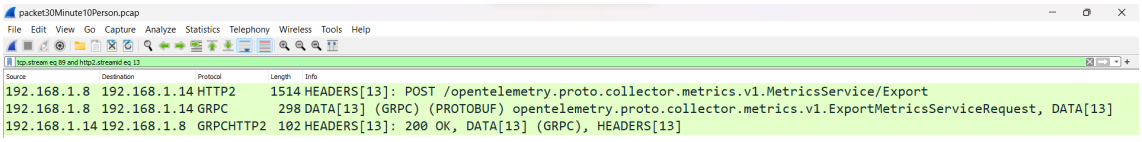
```
syntax = "proto3";

package opentelemetry.proto.common.v1;

message KeyValue {
  string key = 1;
  AnyValue value = 2;
}
```

Výpis 17: Definice souboru .proto ze systému OpenTelemetry pro strukturu klíč-hodnota

Systém OpenTelemetry poskytuje veřejně soubory .proto pro veškerou jejich komunikaci. Najdeme je na úložišti GitHub[3]. Vložíme je do Wiresharku v záložce **Edit** > **Preferences** > **Protocols** > **ProtoBuf**, kde vložíme cestu k souborům a zaklikneme políčko **Load all files**. Poté nastavíme komunikaci na HTTP2 na určitém portu a to pomocí **Analyze** > **Decode As..**, kde přidáme novou položku s hodnotami TCP protokol a port, který je v našem případě 4317 a HTTP2 protokol.



Obrázek 4.1: Ukázka jednoho toku komunikace

```
{
  "name": "process.runtime.dotnet.assemblies.count",
  "description": "The number of .NET assemblies that are currently loaded.",
  "gauge": {
    "data_points": [
      {
        "start_time_unix_nano": "1673739808162277500",
        "time_unix_nano": "1673790268036106100",
        "as_int": "8"
      }
    ]
  }
}
```

Výpis 18: Ukázka metriky ze zachycené komunikace ve Wiresharku za využití mapování na JSON

[3]Repositář s .proto soubory https://github.com/open-telemetry/opentelemetry-proto [12.04.2023]

4.3 Experimenty

Účelem experimentů je nasimulovat nežádoucí stavy v testovacím prostředí, se kterými se můžeme setkat při provozu aplikace. Experimenty budeme provádět na testovacím prostředí, které je popsané v tabulce 4.1.

	Testovací aplikace	**Kolektor + Monitorovací nástroje**
Model	Acer Aspire VN7-572G	Raspberry Pi 4 Model B Rev 1.4
CPU	Intel(R) Core(TM) i5-6200U @ 2.30GHz	Broadcom BCM2711
RAM	8 GiB DDR4	8 GiB LPDDR4

Tabulka 4.1: Konfigurace testovacího prostředí

4.3.1 Experiment č. 1: Výpadek mikroslužby

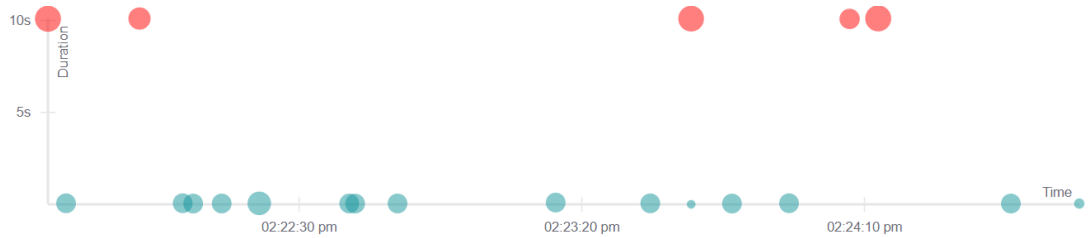
Testovací aplikace je navržena pomocí architektury mikroslužeb, kde každá mikroslužba může běžet na vlastním stroji. K plné funkcionalitě aplikace musí všechny mikroslužby běžet. Tímto experimentem testujeme nedostupnost mikroslužby z důvodu výpadku stroje, na kterém běží, přerušení síťového spojení a mnoho dalších výpadků mikroslužby.

Navození nežádoucího stavu

Nežádoucí stav navodíme vypnutím jedné mikroslužby. Díky využití technologie Docker kontejnerů, kde každá mikroslužba běží ve vlastním kontejneru, vypneme pouze kontejner s danou mikroslužbou. Pro náš experiment vypneme službu `feature-flag-service`.

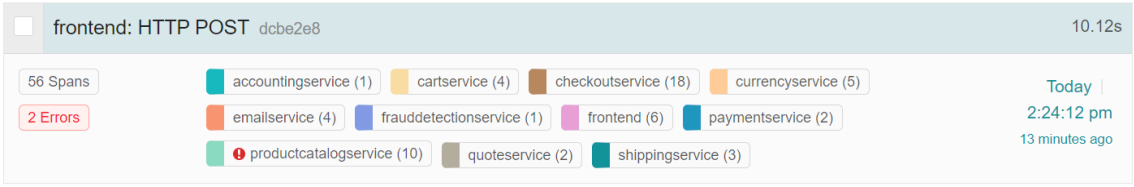
Výsledek

V monitorovacím nástroji Jeager po vybrání mikroslužby, která se snaží o komunikaci s nedostupnou mikroslužbou, hned vidíme v úvodním grafu, že některé stopy obsahují chybovou hlášku.



Obrázek 4.2: Úvodní graf ukazující čas přijatý stopy a její délku trvání

Na obrázku 4.2 vidíme graf s posledními 20 přijatými stopami. Můžeme si všimnout pěti stop, které jsou označeny červenou barvou, která značí stopu s chybou. Nejen podle červené barvy vidíme, že je něco v nepořádku. Délka trvání stopy nám může také pomoci při hledání chyby. Vidíme, že stopy s chybou trvají okolo 10 sekund a zbylé trvají necelou sekundu.



Obrázek 4.3: Stopa obsahující chybu

Na obrázku 4.3 vidíme stopu, ve které se vyskytují dvě chyby a vidíme červený vykřičník u mikroslužby, kde nastaly chyby. Podrobnější informace se dozvíme po otevření detailu stopy.

```
{
  "key": "otel.status_description",
  "type": "string",
  "value":
      "_InactiveRpcError: <_InactiveRpcError of RPC that terminated with:
      tstatus = StatusCode.UNAVAILABLE
      tdetails = failed to connect to all addresses;
      last error: UNKNOWN: ipv4:172.18.0.10:50053:
      Failed to connect to remote host: Connection refused>"
}
```

Výpis 19: Detailní informace o chybě

Na výpisu 19 se nachází pole `status_description`, jež je součástí detailu stopy v monitorovacím nástroji Jeager. Vidíme, že mikroslužba na IP adrese 172.17.0.10 je nedostupná.

4.3.2 Experiment č. 2: Přetížení aplikace v důsledku mnoho aktivních uživatelů

K přetížení dochází, když aplikace zpracovává větší množství požadavků, než je schopna za danou dobu zpracovat. To může nastat z několika důvodů například:

- velký počet aktivních uživatelů,
- aplikace zpracovává velký objem dat,
- aplikace není optimalizována pro svůj účel nebo zpracování dat.

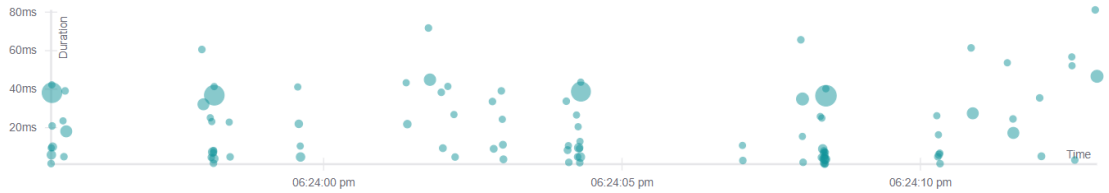
V našem experimentu se budeme věnovat přetížení díky velkému počtu aktivních uživatelů.

Navození nežádoucího stavu

Přetížení nasimulujeme pomocí mikroslužby `load-generator`. V základu simulujeme v testovacím prostředí deset uživatelů a nyní zvýšíme počet na 500.

Výsledek

Přetížení aplikace můžeme vypozorovat již v úvodním grafu se stopami, kde si můžeme všimnout delší doby zpracování požadavků.



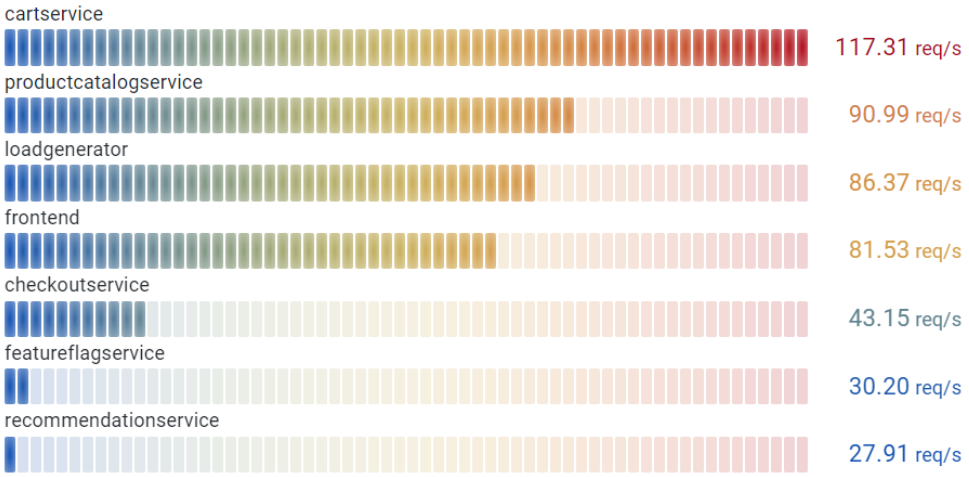
Obrázek 4.4: Úvodní graf délky požadavků u uživatelského rozhraní

Na obrázku 4.4 vidíme požadavky mikroslužby `frontend`, která poskytuje uživatelské rozhraní. Můžeme vidět požadavky trvající maximálně 80 milisekund.

Obrázek 4.5: Úvodní graf délky požadavků při přetížení

Na obrázku 4.5 vidíme, že délka požadavků se rapidně zvýšila. Požadavky nyní trvají až čtyři sekundy. Délka požadavku nemusí nutně znamenat přetížení aplikace. Proto se podíváme i na ostatní telemetrická data.

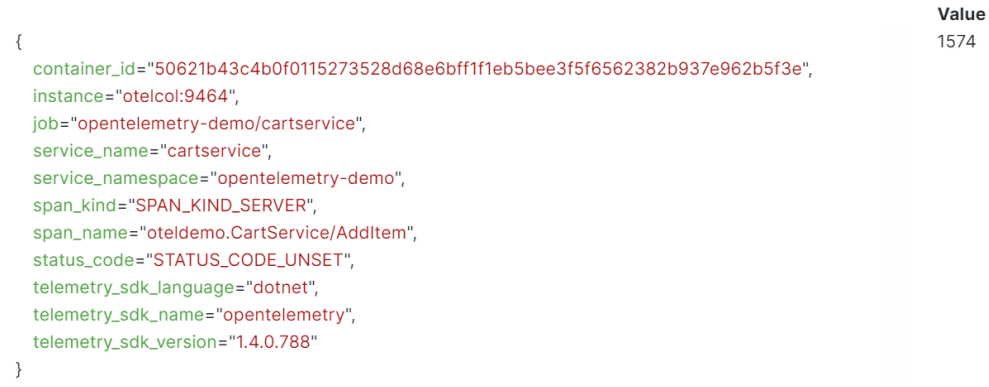
Pro přesnější diagnostikování problému se podíváme na metriky v monitorovacím nástroji Grafana.



Obrázek 4.6: Graf s požadavky na mikroslužby za sekundu

Na obrázku 4.6 vidíme počty požadavků za sekundu na sedm nejvytíženějších služeb testovací aplikace. Díky grafu vidíme, že služba `cartservice` je přetížená. Ostatním službám se také zvedl počet požadavků v normálním poměru. V důsledku toho můžeme určit, že

se s velkou pravděpodobností jedná o přetížení aplikace z důvodu velkého počtu aktivních uživatelů a nejedná se o útok na aplikaci.



Obrázek 4.7: Ukázka detailu počtu požadavku na koncový bod mikroslužby

Na obrázku 4.7 vidíme detailní popis metriky, která představuje počet požadavků na jeden koncový bod služby `cartservice`.

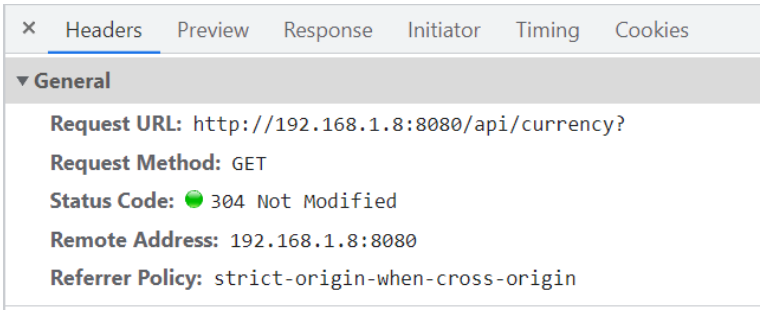
4.3.3 Experiment č. 3: DoS útok na jednu z mikroslužeb

DoS (Denial of Service) je útok na počítačovou síť nebo webový server, při kterém útočník používá jeden počítač k vytížení vybraného zdroje. Pokud by útočník používal více počítačů k útoku jednalo by se o distribuovaný útok DoS (DDoS). Cílem je způsobit výpadek služby nebo její výrazné zpomalení cílového serveru. Hrozí zde ztráta dat, snížení odezvy dokonce výpadek celé služby.

Navození nežádoucího stavu

Útok nasimulujeme pomocí skriptu python, kde využijeme knihovnu `requests`[4] pro odeslání požadavku na testovací aplikaci. Odesílaní jednoho požadavku v nekonečném cyklu by testovací aplikaci moc neohrozilo, proto využijeme také knihovnu `threading`[5], díky které budeme odesílat požadavky ve více vláknech paralelně.

Nejprve se pomocí vývojářského okna v prohlížeči v záložce **Network** podíváme, jaké požadavky se v testovací aplikaci nacházejí a jeden si vybereme.



Obrázek 4.8: Hlavička požadavku z prohlížeče

[4]Dokumentace knihovny requests https://pypi.org/project/requests/ [16.04.2023]
[5]Dokumentace knihovny threading https://docs.python.org/3/library/threading.html [16.04.2023]

Na obrázku 4.8 vidíme hlavičku zachyceného požadavku v prohlížeči, ze které vyčteme informace cílové adresy a o jakou metodu HTTP se jedná.

```
# Funkce k~provedení GET požadavku a výpisu odpovědi
def make_request():
    while True:
        response = requests.get("http://192.168.1.8:8080/api/currency")
        print(response.text)
```

Výpis 20: Funkce pro odeslání požadavku GET

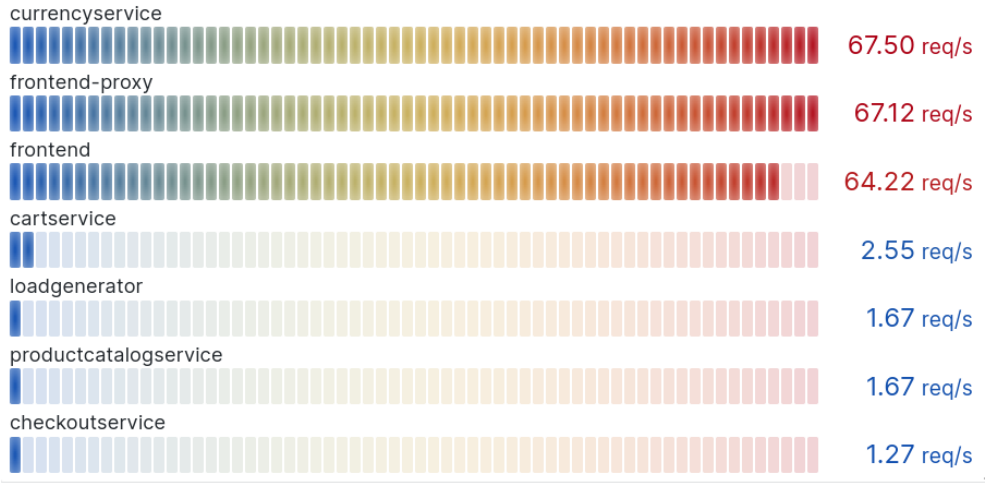
Na výpisu 20 vidíme vytvoření požadavku GET pomocí knihovny `requests` a výpisu odpovědi. Funkci zavoláme ve vytvořených vláknech a vlákna spustíme. Díku tomu docílíme nekonečného paralelního odesílání požadavků na testovací aplikaci.

Výsledek

Než si ukážeme, jaká telemetrická data se nám zobrazí, musíme si nejprve vysvětlit, jak mikroslužby mezi sebou komunikují, abychom později pochopili, jaké mikroslužby jsou přetížené.

Obrázek 4.9: Komunikace mezi mikroslužbami

Na obrázku 4.9 vidíme, že uživatel nekomunikuje s jednotlivými mikroslužbami přímo, ale komunikuje přes `frontend-proxy` a `frontend`. Pokud si tedy pro náš útok vybereme mikroslužbu `currency-service`, budou tím ovlivněny i předchozí dvě mikroslužby.



Obrázek 4.10: Graf s požadavky na mikroslužby za sekundu

Na obrázku 4.10 vidíme vysoké počty požadavků u třech mikroslužeb, které jsou v nepoměru s ostatními službami. Z toho můžeme usoudit, že nejspíš někdo odesílá velký počet požadavků na mikroslužbu `currencyservice` a aplikace se nachází pod DoS útokem. Pokud se chceme dozvědět více informací o útočníkovi, musíme se podívat do detailu stopy.

http.url	`http://192.168.1.8:8080/api/currency`
internal.span.format	`proto`
node_id	
peer.address	`192.168.1.32`
request_size	`0`
response_flags	`-`
response_size	`199`
span.kind	`server`
upstream_cluster	`frontend`
upstream_cluster.name	`frontend`
user_agent	`python-requests/2.28.2`

Obrázek 4.11: Informace o útočníkovi

V monitorovacím nástroji uvidíme velký počet stejných požadavků a po zobrazení detailu požadavku vidíme, že všechny pochází ze stejné adresy. Dozvídáme se i další informace jak vidíme na obrázku 4.11 jako velikost požadavku a odpovědi, použitý nástroj pro odeslání požadavku atd.

4.3.4 Experiment č. 4: Podvržení záznamu Syslog kolektoru

Pomocí telemetrických dat můžeme automatizovat věci na serverech, jako například, vyvažování zátěže nebo nastavení upozornění, když přijde záznam, který bude obsahovat chybu, odešle se správci upozornění. Útočník by tím pádem mohl chtít telemetrická data podvrhnout, aby zmátl daný systém.

Navození nežádoucího stavu

Podvržený záznam Syslog budeme odesílat pomocí skriptu python. Záznamy Syslog se odesílají pomocí protokolu TCP nebo UDP. Pro získání portu, na kterém poslouchá kolektor záznamy Syslog, využijeme nástroj `nmap`. `Nmap` nám umožní detekovat otevřené porty na zadané IP adrese. Pokud se komunikace nachází na standardním portu, `nmap` nám řekne, že se jedná o port pro komunikaci Syslog. Když se komunikace nachází na nestandardním portu, musíme využít nástroj `tcpdump` a zachytit na daných portech komunikaci. Pokud uživatel nepoužívá šifrovanou komunikaci, můžeme ze zachycených paketů přečíst záznam Syslog a zjistit, na jakém portu kolektor naslouchá záznamy.

```python
# Vytvoření TCP schránky
sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
# Připojení ke vzdálenému TCP Syslog serveru
sock.connect(('192.168.1.14', 54527))
# Odesílání Syslog záznamu
sock.send(
    '<13>Apr 19 12:34:56 myhost myprocess: Hello, Syslog!\n'.encode())
# Uzavření schránka
sock.close()
```

Výpis 21: Odeslání podvrženého záznamu na kolektor

Na výpisu 21 vidíme vytvoření schránky TCP a připojení na kolektor a odeslání podvrženého záznamu.

Výsledek

```
2023-04-22 00:12:09 {
                      "body": "<13>Apr 19 12:34:56 myhost myprocess: Hello, s
                    yslog!"
                    }
```

Obrázek 4.12: Podvržený záznam v monitorovacím nástroji

Na obrázku 4.12 vidíme záznam, jež jsme odeslali skriptem mezi ostatními záznamy ze Syslog serveru.

Tomuto útoku můžeme lehce předejít. Syslog přijímač podporuje TSL protokol pro šifrování komunikace. Díky tomu si nikdo odchycené pakety nepřečte, ale ani nedokáže podvrhnout záznam.

4.3.5 Zhodnocení experimentů

Telemetrická data zobrazená v monitorovacích nástrojích odpovídají prováděným experimentům a dali nám přesný pohled co se děje s testovací aplikací.

Kapitola 5

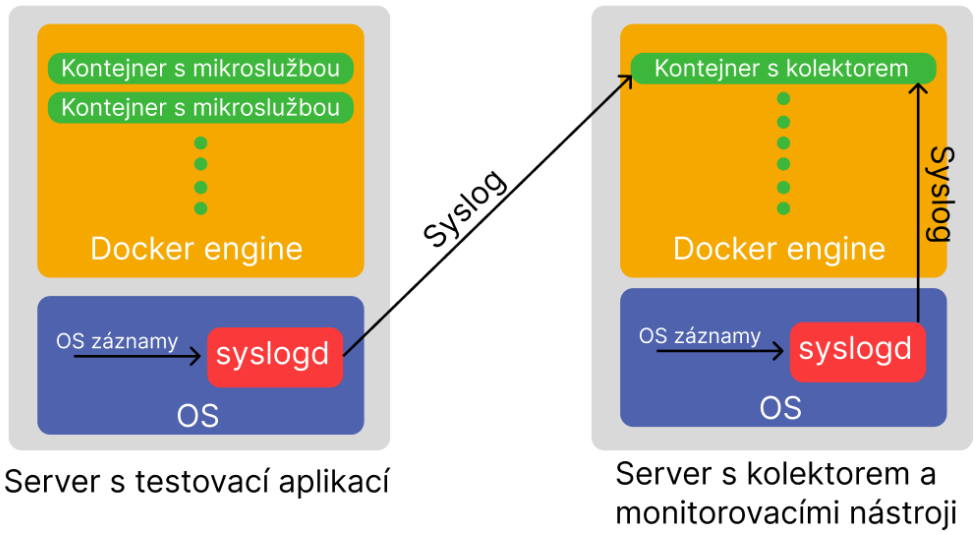
Srovnání systému OpenTelemetry s ostatními monitorovacími systémy

V této kapitole si ukážeme jiné nástroje pro sběr telemetrických dat a ukážeme si odeslání některých telemetrických dat od třetích stran do systému OpenTelemetry. Na konci kapitoly si porovnáme nástroje podle dat a architektury.

5.1 Nástroj pro sběr záznamů Syslog

Syslog [6] je standardizovaný protokol pro přenos záznamů po síti. Původně byl vyvinut pouze pro unixové operační systémy, ale nyní je podporován mnoha operačními systémy, síťovými zařízeními a aplikacemi. Protokol je navržen, aby poskytoval centralizovaný sběr, zpracování a uložení záznamů.

Systém OpenTelemetry umožňuje přijímat záznamy od protokolu Syslog. Podporuje jeho dva formáty záznamu RFC 3164 [6] a RFC 5424 [5]. Odesílání záznamů můžeme zajistit pomocí TCP nebo UDP protokolu. Využíváme rSyslog verze 8.2204.0.



Obrázek 5.1: Schéma odesílání záznamů Syslog

Na obrázku 5.1 vidíme dva servery, kde běží operační systémy s démonem Syslog, který přijímá systémové záznamy (kernel záznamy, mail záznamy, záznamy o přihlášení atd.) a odesílá je na kolektor.

```
*.*  action(type="omfwd"          # pomocí *.* odesíláme všechny záznamy
# Pokud je kolektor nedostupný, vytvoříme soubor s~neodeslanými
# záznamy na disku, aby jsme o~ně nepřišly. Až je kolektor opět dostupný
# záznamy odešleme
queue.filename="fwdRule1"        # unikátní prefix pro soubor na disku
queue.maxdiskspace="1g"          # maximální velikost souboru
queue.saveonshutdown="on"        # uložení souboru při vypnutí
queue.type="LinkedList"          # asynchronní odesílání
action.resumeRetryCount="-1"     # počet pokusů při nedostupnosti kolektoru
                                 # (-1 pro nekonečno)
# Nastavení adresu/doménu, port a protokol pro komunikaci s~kolektorem
Target="192.168.1.14" Port="54526" Protocol="tcp")
```

Výpis 22: Konfigurace aplikace rSyslogu pro odesílání záznamů na kolektor

Ve výpisu 22 vidíme konfiguraci odesílání záznamů na vzdáleného hostitele (kolektor). Soubor s konfigurací nalezneme v `/etc/rSyslog.conf` nebo vytvoříme nový soubor s příponou `*.conf` ve složce `/etc/rSyslog.d`. Využíváme k tomu `omfwd`[1] modul, jež se nachází v základním balíčku rSyslog, takže nemusí být dodatečně instalován a načítán. Můžeme specifikovat, jaké záznamy chceme odesílat.

Po odeslání záznamu musíme na kolektoru použít Syslog přijímač[2], který poskytuje systém OpenTelemetry. Přijímač přidáme v konfiguračním souboru kolektoru, který využívá formát YAML[3] pro ukládaní konfiguračních dat. Přijímač přidáme do položky `receivers`.

```
receivers:
  Syslog/server:
    tcp:
      # přijímá záznamy ze všech IP adres na portu 54526
      listen_address: "0.0.0.0:54526"
    protocol: rfc5424
```

Výpis 23: Konfigurace přijímače kolektoru pro příjem záznamů Syslog

Ve výpisu 23 vidíme nastavení přijímače pro příjem Syslog záznamů verze RFC 5424. Pro přenos záznamů využíváme TCP protokol a posloucháme na portu 54 526. Pro zobrazení využíváme monitorovací nástroj Loki od firmy Grafana, který pro filtrování záznamů

[1] Oficiální dokumentace omfwd modulu https://www.rSyslog.com/files/temp/doc-indent/configuration/modules/omfwd.html [14.03.2023]

[2] Dokumentace Syslog přijímače https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/receiver/Syslogreceiver [14.03.2023]

[3] YAML Ain't Markup Language – formát pro serializaci dat textových souborů https://yaml.org/ [14.03.2023]

vyžaduje štítek (label). Pro přidání štítku využijeme procesor kolektoru a každý záznam označíme štítkem.

```
processors:
  attributes/server:
    actions:
      - action: insert
        key: Syslog
        value: server_eshop
      - action: insert
        key: loki.attribute.labels
        value: Syslog
```

Výpis 24: Konfigurace procesoru kolektoru pro přidání štítku pro filtrování

Ve výpisu 24 vidíme vyžití atribut procesoru[4], kde si prvně vytvoříme atribut s jménem `Syslog` a hodnotou `server_eshop` a poté z něj vytvoříme atribut (štítek) pro monitorovací nastroj Loki. Po přijetí a přiřazení štítku můžeme daný záznam odeslat exportérem na monitorovací nástroj.

```
exporters:
  loki:
    endpoint: "http://loki:3100/loki/api/v1/push"
```

Výpis 25: Konfigurace exportéru kolektoru pro odeslání záznamu na monitorovací nástroj

Ve výpisu 25 vidíme exportér Loki[5], který má definovaný koncový bod, kde monitorovací nástroj očekává záznamy.

```
2023-04-13 22:48:39 {
                      "body": "<30>Apr 13 20:48:39 server dockerd[971]: time=\"2023-04-13T20:4
                    8:39.304233434Z\" level=warning msg=\"[resolver] failed to read from DNS se
                    rver: 127.0.0.53:53, query: ;192.168.1.14.\\tIN\\t AAAA\" error=\"read udp
                    127.0.0.1:36092->127.0.0.53:53: i/o timeout\""
                    }
```

Obrázek 5.2: Ukázka zobrazení záznamu

Na obrázku 5.2 vidíme Syslog záznam, kde celý záznam byl namapován automaticky na pole `body` struktury záznamu OpenTelemetry. Ze záznamu vyčteme, že se jedná o varování pocházející z docker démona o selhání překladu adresy (resolver), který nedokázal číst z DNS serveru.

[4]Dokumentace atribut procesoru https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/processor/attributesprocessor [14.03.2023]

[5]Dokumentace Loki exportéru https://github.com/open-telemetry/opentelemetry-collector-contrib/tree/main/exporter/lokiexporter [14.03.2023]

5.2 Analyzátor síťového provozu NetFlow

NetFlow [4] je síťový protokol, který sbírá a analyzuje data o provozu IP. Správcům sítě poskytuje přehled o síťovém provozu a pomáhá při analýze chování sítě, což pomáhá při optimalizaci výkonu sítě, identifikaci bezpečnostních hrozeb atd.

NetFlow funguje na principu zaznamenávaní síťových toků. Tok je definován jako posloupnost paketů, které mají společnou charakteristiku, jakou jsou stejné zdrojové a cílové IP adresy, čísla portů, protokol a hodnotu priority komunikace (ToS). Tok obsahuje informace také o velikosti přenesených dat, počtu paketů, počáteční a koncový čas toku atd.

Zachytíme toky v síti testovací aplikace pomocí protokolu NetFlow a podíváme se, jaké informace nám poskytnou.

Application	Protocol	Client	Server	Duration	Breakdown	Actual Thpt	Total Bytes	Info
? Unknown	TCP	172.18.0.10 L :42624	192.168.1.14 R :4317	03:01	Client Ser	0 bps —	66.13 KB —	
HTTP DPI	⚠ TCP	172.18.0.20 L :41906	frontend L :http-alt	00:12 sec	Client Server	0 bps ↓	4.56 KB —	frontend:8080/api/produc...
? Unknown	TCP	frontend L :58952	172.18.0.10 L :9555	03:00	Client Server	0 bps —	34.43 KB —	
? Unknown	TCP	172.18.0.19 L :42498	172.18.0.9 L :9092	24:03	Client Server	5.80 kbit/s ↓	1.46 MB ↑	
? Unknown	TCP	172.18.0.18 L :32944	172.18.0.9 L :9092	24:03	Client Server	4.60 kbit/s ↑	1.01 MB ↑	
PostgreSQL Guess	TCP	172.18.0.14 L :52539	172.18.0.3 L :postgresql	24:01	Client Server	1.60 kbit/s ↑	169.01 KB ↑	
HTTP DPI	⚠ TCP	172.18.0.20 L :36312	frontend L :http-alt	00:11 sec	Client Server	0 bps ↓	3.98 KB ↑	frontend:8080/api/produc...

Obrázek 5.3: Ukázka toků z nástroje ntopng

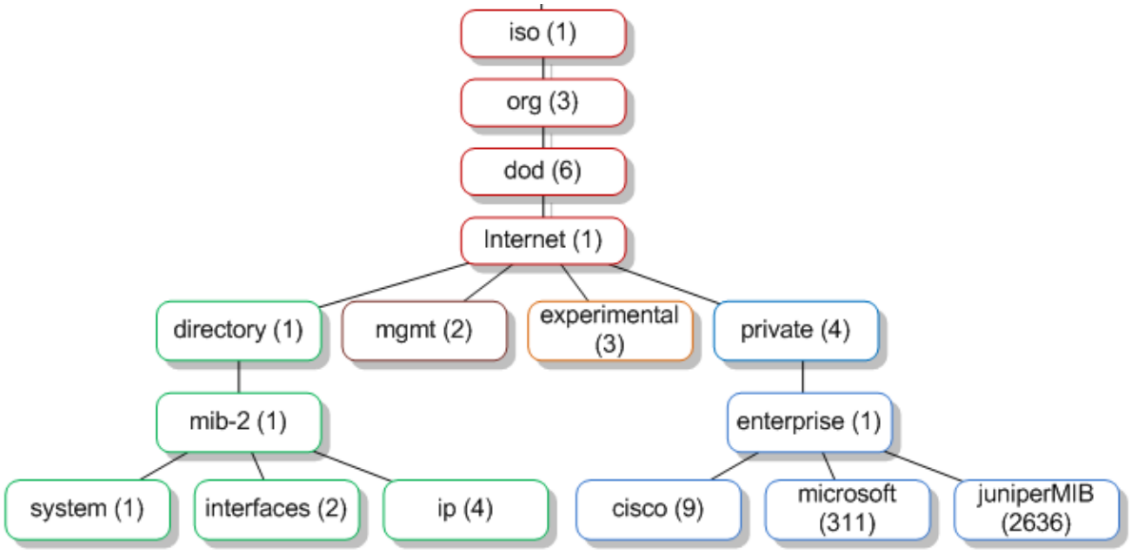
Na obrázku 5.3 vidíme aktuální toky. Můžeme vidět, že NetFlow protokol nám neposkytuje detailní informace o mikroslužbách a zpracování jejich požadavků jako telemetrická data systému OpenTelemtry. Poskytuje nám přehled o síťové komunikaci mezi jednotlivými mikroslužbami nebo komunikaci s kolektorem systému OpenTelemetry.

5.3 Nástroj pro monitorovaní a správu sítě SNMP

SNMP [3] neboli Simple Network Management Protocol je standardizovaný internetový protokol, jenž poskytuje správci sítě správu a monitorování síťových zařízení, jako jsou například směrovače, přepínače, servery a další. Umožňuje správcům sítě spravovat a monitorovat zařízení z jednoho místa. Poskytuje informace o výkonu sítě, využití šířky pásma, ale můžeme sledovat informace i o stavu a konfiguraci jednotlivých síťových zařízení, jako je využití paměti, zatížení CPU nebo verzi bežícího firmwaru.

SNMP funguje na principu manager-agent, kde stanice správce vystupuje jako manager a jednotlivá síťová zařízení jako agenti. Manager odesílá požadavky na agenty, aby získal informace o stavu zařízení.

Data v SNMP jsou reprezentována pomocí Management Information Base (MIB). MIB je databáze obsahující hierarchicky organizované informace o spravovaných zařízeních a sítích. Každý objekt je identifikován pomocí jedinečného čísla (OID). OID se skládá z řady čísel, které jsou odděleny tečkou. Například OID může vypadat takto: 1.3.6.1.1.2.1.



Obrázek 5.4: OID stromová struktura

Na obrázku 5.4 vidíme stromovou strukturu OID. Uzel `system` je reprezentován již zmíněným identifikátorem z ukázky 1.3.6.1.1.2.1.

Ukážeme si, jaká data nám může zobrazit SNMP o testovací aplikaci. Data budeme číst z prvku OID 1.3.6.1.2.1.25.4.2 neboli `hrSWRunEntry`.

Název	**OID**	**Popis**
hrSWRunIndex	1.3.6.1.2.1.25.4.2.1.1	Unikátní hodnota programového vybavení
hrSWRunName	1.3.6.1.2.1.25.4.2.1.2	Název programového vybavení
hrSWRunID	1.3.6.1.2.1.25.4.2.1.3	ID produktu běžícího programu
hrSWRunPath	1.3.6.1.2.1.25.4.2.1.4	Umístění na disku hostitele
hrSWRunParameters	1.3.6.1.2.1.25.4.2.1.5	Parametry, které byly dodány při prvním načtení
hrSWRunType	1.3.6.1.2.1.25.4.2.1.6	Typ programu: 'unknown': 1, 'operatingSystem': 2, 'deviceDriver': 3, 'application': 4.
hrSWRunStatus	1.3.6.1.2.1.25.4.2.1.7	Status o běhu programu: 'running': 1, 'runnable': 2, 'notRunnable': 3, 'invalid': 4.

Tabulka 5.1: Potomky prvku hrSWRunEntry

Pro čtení SNMP dat využijeme nástroj snmpwalk[6]. Nejprve si vypíšeme názvy programového vybavení hostitele a podíváme se na programy související s testovací aplikací.

```
snmpwalk -v2c -c public localhost 1.3.6.1.2.1.25.4.2.1.2
```

Výpis 26: Výpis hrSWRunName pomocí nástroje snmpwalk

[6]Manuál nástroje snmpwalk https://linux.die.net/man/1/snmpwalk [16.04.2023]

Pomocí příkazu ve výpisu 26 si vypíšeme všechny názvy programového vybavení na hostiteli. Jelikož výsledek je velmi dlouhý, ukážeme si jen část, kde se budou nacházet položky, tykající se testovací aplikace.

```
iso.3.6.1.2.1.25.4.2.1.2.2931 = STRING: "docker-proxy"
iso.3.6.1.2.1.25.4.2.1.2.2985 = STRING: "shippingservice"
iso.3.6.1.2.1.25.4.2.1.2.3813 = STRING: "locust"
iso.3.6.1.2.1.25.4.2.1.2.3820 = STRING: "productcatalogs"
```

Výpis 27: Část výsledku příkazu pro výpis hrSWRunName

Na výpisu 27 vidíme `docker-proxy`, které je zodpovědné za probíhající komunikaci mezi kontejnery a hostitelem. Dále vidíme tři mikroslužby. V následujících tabulkách si ukážeme informace, které nám SNMP poskytne o jedné mikroslužbě a docker-proxy. Na jednotlivé informace se budeme dotazovat na konkrétní programové vybavení a to docílíme tím, když OID rozšíříme o poslední číslo z výsledku. Například pro dotazování na docker-proxy pro zjištění hrSWRunIndex použijeme OID 1.3.6.1.2.1.25.4.2.1.1.2931.

	docker-proxy	**shippingservice**
hrSWRunIndex	2931	2985
hrSWRunName	docker-proxy	shippingservice
hrSWRunID	ccitt.0	ccitt.0
hrSWRunPath	/usr/bin/docker-proxy	/app/shippingservice
hrSWRunParameters	-proto tcp -host-ip :: -host-port 9001 -container-ip 172.18.0.6 -container-port 9001	
hrSWRunType	4	4
hrSWRunStatus	2	2

Tabulka 5.2: Informace ze SNMP

Z tabulky 5.2 vidíme, že se dozvídáme pouze informace o kontejneru, jako je IP adresa a port a název mikroslužby, a zda běží. SNMP nám neříká žádné informace o mikroslužbě samotné jako například její vytížení, odezvu, komunikaci atd.

5.4 Srovnání jiných nástrojů se systémem OpenTelemetry

V této podkapitole si srovnáme zmíněné nástroje se systémem OpenTelemetry a porovnáme je podle architektury a zda by nám byly schopny ukázat nežádoucí stavy z experimentů.

5.4.1 Srovnání podle architektury

Architekturu systému OpenTelemetry jsme si popsali ve druhé kapitole 2.4, tak si ukážeme pouze architekturu nástrojů zmíněných v této kapitole.

Architektura Syslog

Démon Syslog běží na pozadí a přijímá záznamy. Může přijímat záznamy od procesu, služby nebo aplikace bežící na počítači, která generuje záznamy o svém běhu. Záznamy ukládá v textových souborech nebo odesílá na server Syslog.

Architektura NetFlow

Architektura se skládá ze tří prvků exportér, kolektor a analyzátor. Exportér vložíme do síťové komunikace jako nezávislou sondu, která bude poslouchat komunikaci, vytvářet toky a odesílat je na kolektor. Kolektor ukládá jednotlivé toky pro analyzátor. Pomocí analyzátoru studujeme jednotlivé toky a získáváme z nich užitečné informace.

Architektura SNMP

Architektura SNMP obsahuje dva prvky manažera a agenty. Agent představuje monitorovaný prvek a manažer je stanice, která odesílá požadavky na agenty a získává data.

5.4.2 Srovnání podle dat

Porovnáme zmíněné nástroje, zda by nám byly schopny detekovat nežádoucí stavy navozené experimenty.

Nástroj Syslog

Syslog by nám nebyl schopný detekovat ani jeden z experimentů, jelikož v našem testovacím prostředí nevidí do mikroslužeb. Jediný experiment, který by jsme mohli odhalit, je podvržení záznamu Syslog. Kdyby správce porovnal záznamy uložené na serveru se záznamy v monitorovacím nástroji, mohl by si všimnout, že v monitorovacím nástroji se nachází záznamy, které nejsou uloženy na serveru.

Analyzátor NetFlow

NetFlow nám umožní lehce detekovat dva experimenty a to přetížení pomocí mnoho aktivních uživatelů a DoS útok. Uvidíme veliké množství nových toků. Experimenty rozlišíme od sebe pomocí zdrojové IP adresy v toku, kde DoS útok bude mít všude stejnou adresu a přetížení bude obsahovat rozdílné adresy. Výpadek a podvržení poznáme trochu obtížněji. Musíme pozorovat toky a mít znalost, které toky se běžně objevují. U podvrženého záznamu uvidíme tok, který obsahuje cílovou adresu kolektoru systému OpenTelemetry a zdrojová adresa nebude serveru, kde běží testovací aplikace. Výpadek mikroslužby poznáme, že toky se přestanou objevovat.

Nástroj SNMP

Nástroj SNMP nám pomůže při detekci prvních třech experimentů. Protože nám poskytuje informaci akorát o tom zda mikroslužba běží nebo ne. Nepoznáme, ale z jakého důvodu mikroslužba neběží.

5.4.3 Závěrečné srovnání

Vidíme, že představené nástroje v kapitole pět nám nedávají moc přehled o dění v mikroslužbách a nemůžeme pomocí nich většinou detekovat jejich chování, ale je to i dáno jejich architekturou, protože jediný systém OpenTelemetry běží uvnitř mikroslužby a tak je nám schopný poskytovat informace o vnitřním chodu.

Kapitola 6

Závěr

Cílem práce bylo prozkoumat nový systém OpenTelemetry, jež má sloužit jako jednotný nástroj pro sběr telemetrických dat. Nástroj je zaměřen na sběr dat z cloudových aplikací. Proto nemůžeme říct, že budeme využít pouze systém OpenTelemetry pro monitorování celé infrastruktury. Jedná se spíše o komplement k zmíněným nástrojům, protože například o strojích, na kterých aplikace běží by nám poskytl minimální informace.

Ze začátku práce jsem musel nastudovat, co je systém OpenTelemetry a jak funguje. Pro testovací prostředí jsem si vybral demo aplikaci, kterou poskytují samotní tvůrci. Prostředí jsem rozdělil na dvě části, aby mohly běžet nezávisle na dvou serverech. Jedná část obsahuje webový obchod a druhá část je kolektor s monitorovacími nástroji. Dále jsem vložil do testovacího prostředí jiné nástroje a pozoroval jaké data nám poskytnout. Poté jsem prováděl experimenty nad prostředím a pozoroval jsem, zda budou telemetrická data odpovídat danému nežádoucímu stavu.

Jedná se o nový standard, který se stále vyvíjí. V průběhu práce se stalo, že se třikrát změnila konfigurace jednoho z procesorů v kolektoru a chybí podpora generování některých telemetrických dat u určitých technologií, převážně se jedná o záznamy. Přesto tento nástroj hodnotím velmi kladně. Jako jediný nástroj ze zmíněných v práci, nám poskytl přehled o vnitřním chodu mikroslužeb. Díky němu můžeme vidět počty požadavků na jednotlivé koncové body mikroslužeb, jejich odezvy, přerušení spojení mezi mikroslužbami. Poskytuje nám i detailní informace o jednotlivých požadavcích. Z požadavku můžeme udělat stopu, jež se skládá z více operacích, pomocí automatického generování telemetrických dat. Pokud nám nestačí automatické rozdělení požadavku, můžeme ho manuálně rozdělit na operace a podívat se, která část tvrdá nejdéle a jaké informace obsahují.

S největším problémem v této bakalářské práci jsem se setkal u čtení dat z paketu. Nemohl jsem najít správné soubory .proto, pomocí kterých převádíme serializovaná binární data na čitelná pro člověka.

Největší přínos tato bakalářská práce poskytne lidem, kteří se rozhodují využít systému OpenTelemetry. Poskytne jim informace jak generovat telemetrická data z aplikace, jak je sbírat a zobrazovat v monitorovacím nástroji.

Literatura

[1] Anand, A. *OpenTelemetry Collector – architecture and configuration guide* [online]. 2022 [cit. 2022-12-03]. Dostupné z: https://signoz.io/blog/opentelemetry-collector-complete-guide.

[2] Anand, A. *OpenTelemetry Logs – A Complete Introduction and Implementation* [online]. 2022 [cit. 2022-11-25]. Dostupné z: https://signoz.io/blog/opentelemetry-logs.

[3] Case, J. D., Fedor, M., Schoffstall, M. L. a Davin, J. R. *Simple Network Management Protocol (SNMP)* [Internet Requests for Comments]. STD 15. RFC Editor, May 1990 [cit. 2022-04-20]. http://www.rfc-editor.org/rfc/rfc1157.txt.

[4] Claise, B. *Cisco Systems NetFlow Services Export Version 9* [Internet Requests for Comments]. RFC 3954. RFC Editor, October 2004 [cit. 2022-04-20]. http://www.rfc-editor.org/rfc/rfc3954.txt.

[5] Gerhards, R. *The Syslog Protocol* [Internet Requests for Comments]. RFC 5424. RFC Editor, March 2009 [cit. 2022-04-14]. http://www.rfc-editor.org/rfc/rfc5424.txt.

[6] Lonvick, C. *The BSD Syslog Protocol* [Internet Requests for Comments]. RFC 3164. RFC Editor, August 2001 [cit. 2022-04-14]. http://www.rfc-editor.org/rfc/rfc3164.txt.

[7] OpenTelemetry. *Logs Data Model* [online]. 2020 [cit. 2022-03-14]. Dostupné z: https://github.com/open-telemetry/opentelemetry-specification/blob/main/specification/logs/data-model.md.

[8] OpenTelemetry. *Baggage* [online]. 2022 [cit. 2022-11-20]. Dostupné z: https://opentelemetry.io/docs/concepts/signals/baggage/.

[9] OpenTelemetry. *Logs* [online]. 2022 [cit. 2022-11-20]. Dostupné z: https://opentelemetry.io/docs/concepts/signals/logs/.

[10] OpenTelemetry. *Metrics* [online]. 2022 [cit. 2022-11-20]. Dostupné z: https://opentelemetry.io/docs/concepts/signals/metrics/.

[11] OpenTelemetry. *OpenTelemetry Protocol Specification* [online]. 2022 [cit. 2022-12-24]. Dostupné z: https://opentelemetry.io/docs/reference/specification/protocol/otlp/.

[12] OpenTelemetry. *Traces* [online]. 2022 [cit. 2022-11-15]. Dostupné z: https://opentelemetry.io/docs/concepts/signals/traces/.

Příloha A

Ukázka telemetrických dat

V této kapitole si ukážeme program, kde nejprve zachytíme telemetrická data pomocí automatického sběru. Poté ho přepíšeme pro manuální sběr a zachytíme telemetrická data.

Aplikace představuje server s jedním koncovým bodem `/hello_world`, který po obdržení HTTP GET požadavku, odešle odpověď s pozdravem.

Automatický sběr dat

Zdrojový kód

```python
from flask import Flask, request

app = Flask(__name__)


@app.route("/hello_world")
def server_request():
    print(request.args.get("param"))
    return "Hi"


if __name__ == "__main__":
    app.run(port=8080)
```

Výpis 28: Ukázka kódu pro automatický sběr dat

Vygenerovaná data


```
{
    "name": "/hello_world",
    "context": {
        "trace_id": "0x52febf311696be92218395ff71e5222d",
        "span_id": "0x2b17fe881b521fba",
        "trace_state": "[]"
    },
    "kind": "SpanKind.SERVER",
    "parent_id": "0x82603ff759ca869a",
    "start_time": "2023-01-15T10:59:44.449634Z",
    "end_time": "2023-01-15T10:59:44.451575Z",
    "status": {
        "status_code": "UNSET"
    },
    "attributes": {
        "http.method": "GET",
        "http.server_name": "127.0.0.1",
        "http.scheme": "http",
        "net.host.port": 8080,
        "http.host": "localhost:8080",
        "http.target": "/hello_world?param=Hello",
        "net.peer.ip": "127.0.0.1",
        "http.user_agent": "python-requests/2.28.2",
        "net.peer.port": 41678,
        "http.flavor": "1.1",
        "http.route": "/hello_world",
        "http.status_code": 200
    },
    "events": [],
    "links": [],
    "resource": {
        "attributes": {
            "telemetry.sdk.language": "python",
            "telemetry.sdk.name": "opentelemetry",
            "telemetry.sdk.version": "1.15.0",
            "telemetry.auto.version": "0.36b0",
            "service.name": "unknown_service"
        },
        "schema_url": ""
    }
}
```


Výpis 29: Výsledná data automatického sběru dat

Manuální sběr dat

Zdrojový kód

```
from flask import Flask, request
from opentelemetry import trace
from opentelemetry.instrumentation.wsgi import collect_request_attributes
from opentelemetry.propagate import extract
from opentelemetry.sdk.trace import TracerProvider
from opentelemetry.sdk.trace.export import (
    BatchSpanProcessor,
    ConsoleSpanExporter,
)

app = Flask(__name__)

trace.set_tracer_provider(TracerProvider())
tracer = trace.get_tracer_provider().get_tracer(__name__)

trace.get_tracer_provider().add_span_processor(
    BatchSpanProcessor(ConsoleSpanExporter())
)


@app.route("/hello_world")
def server_request():
    with tracer.start_as_current_span(
        "hello_world",
        context=extract(request.headers),
        kind=trace.SpanKind.SERVER,
        attributes=collect_request_attributes(request.environ),
    ):
        print(request.args.get("param"))
        return "Hi"


if __name__ == "__main__":
    app.run(port=8080)
```

Výpis 30: Ukázka kódu pro manuální sběr dat

Vygenerovaná data


```
{
    "name": "hello_world",
    "context": {
        "trace_id": "0x9ebd0f920174b3484227b18f08a246d4",
        "span_id": "0x58534f4ca1ee43b0",
        "trace_state": "[]"
    },
    "kind": "SpanKind.SERVER",
    "parent_id": "0x9f2996581d1b9d34",
    "start_time": "2023-01-15T10:57:43.432127Z",
    "end_time": "2023-01-15T10:57:43.432179Z",
    "status": {
        "status_code": "UNSET"
    },
    "attributes": {
        "http.method": "GET",
        "http.server_name": "127.0.0.1",
        "http.scheme": "http",
        "net.host.port": 8080,
        "http.host": "localhost:8080",
        "http.target": "/hello_world?param=Hello",
        "net.peer.ip": "127.0.0.1",
        "http.user_agent": "python-requests/2.28.2",
        "net.peer.port": 36652,
        "http.flavor": "1.1"
    },
    "events": [],
    "links": [],
    "resource": {
        "attributes": {
            "telemetry.sdk.language": "python",
            "telemetry.sdk.name": "opentelemetry",
            "telemetry.sdk.version": "1.15.0",
            "service.name": "unknown_service"
        },
        "schema_url": ""
    }
}
```


Výpis 31: Výsledná data manuálního sběru dat

Příloha B

Obsah přiloženého paměťového média

Přiložené paměťové médium obsahuje následující adresářovou strukturu:

- `/`
  - `bp/` — Složka se soubory pro překlad textu bakalářské práce
  - `experiments/` — Složka s experimenty
    - `01/`
      - `01.sh` — Skript s experimentem 1
    - `03/`
      - `03.py` — Skript s experimentem 3
    - `04/`
      - `04.py` — Skript s experimentem 4
  - `opentelemetry-demo/`
    - `src/` — Složka se zdrojovými kódy mikroslužeb
    - `.env` — Konfigurační soubor
    - `docker-compose.yml` — Compose soubor s kontejnery
  - `otel-collector/`
    - `src/` — Složka se soubory pro kolektor a monitorovací nástroje
    - `.env` — Konfigurační soubor
    - `docker-compose.yml` — Compose soubor s kontejnery
  - `captureData.sh` — Skript pro sběr paketů v kapitole 3.2
  - `packetsOTLP.pcap` — Soubor s pakety protokolu OTLP
  - `README.md` — Soubor popisující práci s testovacím prostředím
  - `thesis.pdf` — Bakalářská práce ve formátu PDF