Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/09BE21Marjanovic

УПОТРЕБА ZIO TELEMETRY БИБЛИОТЕКЕ ЗА ПРИКУПЉАЊЕ ДИСТРИБУИРАНИХ ТРАГОВА У СИСТЕМИМА БАЗИРАНИМ НА МИКРОСЕРВИСНОЈ АРХИТЕКТУРИ

USING ZIO TELEMETRY FOR COLLECTING DISTRIBUTED TRACES IN MICROSERVICE-BASED SYSTEMS

Драгутин Марјановић, *Факултет техничких наука, Нови Сад*

Област – СОФТВЕРСКО ИНЖЕЊЕРСТВО И ИНФОРМАЦИОНЕ ТЕХНОЛОГИЈЕ

Кратак садржај – *У овом раду приказан је дистрибуирани систем базиран на микросервисној архитектури који, уз коришћење ZIO Telemetry библиотеке, има способност прикупљања и слања дистрибуираних трагова на екстерни алат за анализу.*

Кључне речи: *дистрибуирано праћење, телеметрија, микросервиси, дистрибуирани системи*

Abstract – *This paper presents a distributed system based on a microservice architecture that, with the use of the ZIO Telemetry library, has the ability to collect and send distributed traces to an external analysis tool.*

Keywords: *distributed tracing, telemetry, microservices, distributed systems*

1. УВОД

У данашње вријеме, архитектура модерних софтверских система најчешће се ослања на микросервисну и *serverless* архитектуру, а неријетко и на екстерне (енг. *third-party*) софтверске компоненте изграђене у сличном маниру. Разлог томе је што тако описана архитектура доноси бројне бенефите при развоју апликација. С друге стране, употреба ових архитектура уноси комплексност при прегледности (енг. *visibility*) система. Разумијевање понашања комплексних дистрибуираних система захтијева посматрање повезаних активности кроз више различитих програма и машина [1].

Сходно томе, јавља се потреба за техникама и алатима за посматрање (енг. *observability*) и надгледање (енг. *monitoring*) сложених дистрибуираних система.

Најчешћи механизми коришћени за посматрање система су – логови (енг. *logs*) и метрике (енг. *metrics*) који имају два битна ограничења – унапријед се дефинише шта се прати и информације које се прате се налазе у оквиру једне компоненте (енг. *component-based*) [2]. Да би се превазишла ова ограничења, развијена је техника дистрибуираног праћења (енг. *distributed tracing*).

Циљ овог рада јесте да читаоцу пружи теоријске основе технике дистрибуираног праћења, као и да прикаже важност њене употребе у дистрибуираним системима базираним на микросервисној архитектури.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Игор Дејановић, ванр. проф.

2. ТЕЛЕМЕТРИЈА

Појам телеметрије у софтверском инжењерству односи се на прикупљање података: метрика, логова и дистрибуираних трагова, неопходних за посматрање и анализу система.

2.1 Метрике

Метрике представљају било какве мјерљиве податке које бисмо жељели да пратимо. Издвајају се два типа метрика – инфраструктурне (енг. *platform metrics*) које се односе на саму инфраструктуру система и служе за евалуацију перформанси система на ниском нивоу, а представљају и значајне показатеље доступности (енг. *availability*) и апликативне (енг. *application metrics*) које су везане за саму логику апликације и корисне су за развојне тимове при разумијевању рада самог система са аспекта његове употребе.

2.2 Логови

Логови представљају поруке које систем генерише када се догоди специфичан догађај у систему. Препоручена пракса је да се логови структуирају на конзистентан начин како би се олакшала њихова централизација. Циљ структуираног логовања јесте увођење формата који олакшава процесирање помоћу рачунара (енг. *machine-readable*). Тако структуирани, погодни су за филтерисање, претрагу и дефинисање међусобне повезаности (корелација логова), али и за даље процесирање у домену аналитике пословне интелигенције (енг. *business intelligence*).

2.3 Дистрибуирано праћење

Дистрибуирано праћење обухвата технике праћења/посматрања апликација, посебно оних који користе микросервисну архитектуру. За имплементацију технике неопходно је извршити инструментацију изворног кода. Инструментација представља процес измјене изворног програмског кода у циљу додавања тачака за праћење (енг. *trace points*). Основни концепти дистрибуираног праћења су:

- дистрибуирани траг (енг. *trace*) – приказује путању захтјева кроз скуп компоненти неопходних за његову обраду.
- *span* – представља означени временски интервал који се дефинише као „јединица рада” (енг. *unit of work*) чија је грануларност најчешће величине *RPC* позива или неког другог међупроцесног (енг. *cross-process*) позива.

• пропагација контекста – задужена за повезивање активности између различитих компоненти система.

3. СТАНДАРДИ

Како је дистрибуирано праћење добијало на важности у продукционим системима, појавила се потреба дефинисања стандарда у области чиме се рјешавају проблеми као што су прикупљање дистрибуираних трагова, пропагација контекста, итд.

У наставку поглавља биће описана три најзначајнија стандарда у овом тренутку, а то су – *OpenCensus*, *OpenTracing* и *OpenTelemetry*.

3.1 OpenCensus

Представља скуп библиотека за различите програмске језике које омогућавају прикупљање метрика и дистрибуираних трагова, те слање података алатима за анализу по избору у реалном времену [3].

У *OpenCensus* стандарду дистрибуирани траг се идентификује јединственим низом од 16 бајтова који се зове *TraceID*. Слично као и трагови, *span*-ови се идентификују јединственим низом од 8 насумично генерисаних бајтова што представља *SpanID*.

Горе наведени идентификатори (*TraceID* и *SpanID*), уз опционе метаподатке чине *SpanContext*. Унутар истог процеса, контекст се прослеђује у тзв. контекст објекту. Приликом комуникације између процеса, контекст се серијализује у заглавља протокола.

3.2 OpenTracing

Састоји се од *API* спецификације, радних оквира (енг. *framework*) и библиотека које су имплементирале спецификацију, као и од саме документације везане за пројекат [4].

За разлику од *OpenCensus* стандарда који пружа подршку за прикупљање и дистрибуираних трагова и метрика, *OpenTracing* омогућава само прикупљање дистрибуираних трагова.

3.3 OpenTelemetry

OpenTelemetry обједињује *OpenCensus* и *OpenTracing* пројекте узимајући најбитније карактеристике од оба и уклањајући њихове појединачне недостатке.

У суштини, представља интегрисан скуп *API*-ја и библиотека, као и механизам за прикупљање (енг. *collection mechanism*) заједно са агентом (енг. *agent*) и компонентом за прикупљање (енг. *collector*). Укључује једноставну пропагацију контекста, дистрибуиране трагове, метрике, итд. За сваки језик нуди јединствен скуп *API*-ја, библиотека као и спецификацију формата података телеметрије (енг. *data specifications*) [5].

4. МИКРОСЕРВИСИ

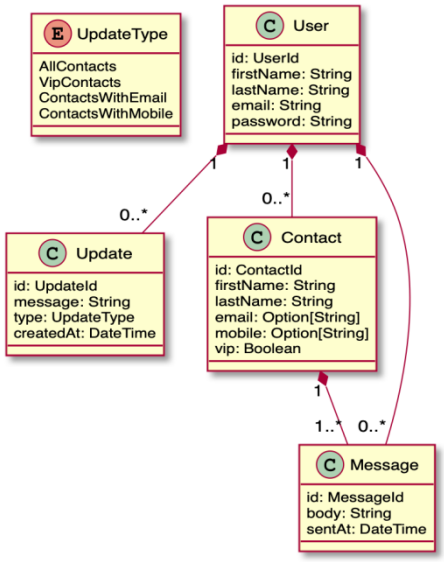
Микросервисна архитектура представља приступ у софтверској архитектури у ком се апликација моделује као скуп аутономних, доменски дефинисаних сервиса који се извршавају у изолованим процесима, а међусобно комуницирају користећи синхроне и асинхроне комуникационе протоколе [6].

Надгледање система базираних на микросервисној архитектури представља тежак задатак из разлога што традиционални алати фокусирани на само једну машину (енг. *machine-centric*) нису ефикасни [7]. Како су микросервиси дистрибуирани по природи они имају другачије захтјеве за надгледање (енг. *monitoring requirements*). Слично као и код метрика, проблем логова у микросервисима је то што нам говоре само о једној инстанци сервиса. Процес надгледања компоненти микросервисне архитектуре започећемо једноставном идејом – надгледаћемо сваку компоненту засебно, а затим агрегирати податке да бисмо добили ширу слику. Агрегација захтијева повезивање података (енг. *correlating data*) из различитих сервиса што омогућава праћење захтјева неопходних за идентификацију неправилности.

Наравно, надгледање дистрибуираних система заснованих на микросервисној архитектури треба да се бави цјелокупном инфраструктуром система, а не само компонентама које се развијају. То значи да је неопходно обезбиједити подршку и за надгледање складишта података, подсистема за комуникацију и размјену порука, итд.

5. СПЕЦИФИКАЦИЈА СИСТЕМА

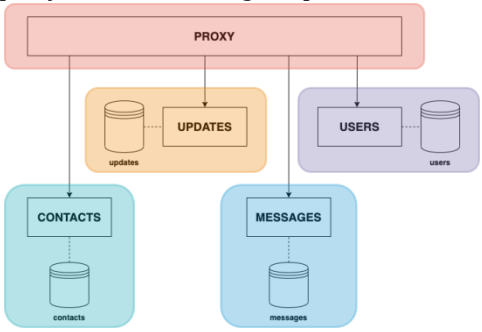
У овом раду дефинисан је систем за аутоматско слање порука контактима који се налазе у корисничком електронском именику. У систему се издвајају доменски концепти: корисник (енг. *user*), контакт (енг. *contact*), порука (енг. *message*) и порука намијењена групи контаката (енг. *update*). Дијаграм приказан на *слици 5.1.* пружа њихов детаљнији приказ, те логичке везе између њих.



Слика 5.1. *Дијаграм домена система*

5.1 Архитектура рјешења

На *слици 5.1.1.* дат је шематски приказ архитектуре који приказује систем базиран на миркосервисној архитектури и састоји се од пет аутономних сервиса, а то су: *proxy*, *contacts*, *messages*, *updates* и *users*.



Слика 5.1.1. *Шематски приказ архитектуре система*

6. ИМПЛЕМЕНТАЦИЈА РЈЕШЕЊА

У претходном поглављу дата је спецификација система. У овом поглављу позабавићемо се основним алатима на које се наслања имплементација горе предложеног рјешења – *ZIO Telemetry* библиотеком и *Jaeger* алатом. Такође, биће истакнути и најзначајнији исјечци програмског кода неопходни за обезбјеђивање инструментације у циљу прикупљања најбитнијих дистрибуираних трагова.

6.1 ZIO Telemetry библиотека

Да би се олакшало функционално програмирање у Скала (енг. *Scala*) програмском језику настао је *ZIO* уз који се развија и цио екосистем намијењен у ту сврху. Језгро библиотеке чини ZIO, моћан тип ефекта, инспирисан Хаскеловим (енг. *Haskell*) IO монадом, приказан на *листингу 6.1.1* [8].

```
ZIO[-R, +E, +A] = R => Either[E, A]
```

Листинг 6.1.1. *Опис ZIO ефекта*

У суштини, ZIO се дефинише као опис израчунавања (енг. *computation*) које захтијева окружење (енг. *environment*), и може резултирати грешком E или се извршити успјешно вриједношћу типа A.

Као дио богатог екосистема развијена је и *ZIO Telemetry* библиотека која представља имплементацију *OpenTracing* клијента наслањајући се на *ZIO*, за чије је коришћење неопходно обезбиједити Clock и OpenTelemetry сервисе у ZIO окружењу.

6.2 Jaeger алат

Jaeger је софтверско рјешење дистрибуираног система за праћење, отвореног кода (енг. *open source*), имплементирано у програмском језику Го (енг. *Go*) ослањајући се на већ постојећа рјешења као што су *Dapper* и *Zipkin* [9]. Основне карактеристике су му: дистрибуирана пропагација контекста, анализа међусервисне зависности, утврђивање узрока отказа (енг. *root cause analysis*), посматрање дистрибуираних трансакција, итд. Такође, компатибилан је са *OpenTracing* стандардом описаном у *секцији 3.2*.

Алат пружа моћан графички кориснички интерфејс (енг. *GUI*) који на недвосмислен начин приказује прикупљене дистрибуиране трагове гдје је нежељено понашање јасно уочљиво. Поред тога, пружа подршку за генерисање усмјереног графа (енг. *DAG*) сервисне међузависности, што представља значајан бенефит у системима који имају на стотине микросервиса.

6.3 Инструментација изворног програмског кода

Приликом имплементације предложеног система, неопходно је обезбиједити инструментацију кода у циљу добијања дистрибуираних трагова. У наставку су приказани најзначајнији исјечци (енг. *snippet*) кода коришћени у ту сврху.

Основни градивни елемент сваког дистрибуираног трага јесте *span*.

```
for {
  span <- ZIO.accessM[AppEnv](_.telemetry.root("OP-NAME"))
  ...
  _    = span.finish()
} yield res
```

Листинг 6.3.1. *Шаблон за креирање коријенског span-а*

У основи, постоје два начина за креирање *span*-а – коријенски *span* (*листинг 6.3.1*) и *span* који садржи референцу на родитељски *span* (*листинг 6.3.2*).

```
val headers = req.headers.toList.map(h => h.name.value -> h.value).toMap
for {
  span <- ZIO.accessM[AppEnv](
          _.telemetry.spanFrom[TextMap](
            HTTP_HEADERS,
            new TextMapAdapter(headers.asJava),
            "OP-NAME"
          )
        )
  ...
  _    = span.finish()
} yield res
```

Листинг 6.3.2. *Шаблон за креирање span-а на основу заглавља HTTP захтјева*

Серијализација контекста у заглавља у сервисима нашег система врши се као што је приказано у *листингу 6.3.3*.

```
buffer  <- UIO.succeed(new TextMapAdapter(mutable.Map.empty[String,
String].asJava))
_       <- OpenTracing.inject(HTTP_HEADERS, buffer)
headers <- extractHeaders(buffer)
...
def extractTracingHeaders(adapter: TextMapAdapter) = {
  val m = mutable.Map.empty[String, String]
  UIO(adapter.forEach { entry =>
    m.put(entry.getKey, entry.getValue)
    ()
  }).as(m.toMap)
}
```

Листинг 6.3.3. *Шаблон за серијализацију контекста у заглавље захтјева*

Да бисмо проширили *span* додатним контекстним информацијама користимо ознаке и логове. Њихова суштинска разлика је да се ознаке односе на цио *span*, док се логови односе на специфичне догађаје који су се десили током извршавања самог *span*-а. *Листинг 6.3.4.* приказује поступак додавања ознака и логова.

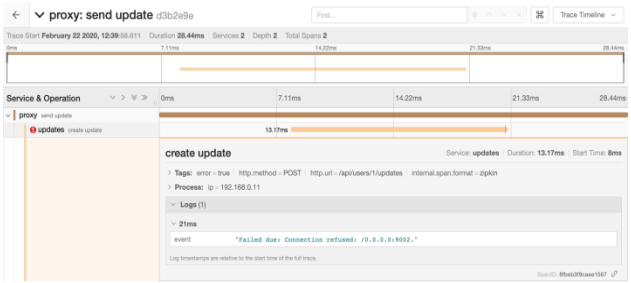
```
for {
  span  <- ...
  _     = span.setTag(TagName, TagValue)
  _     = span.log(TimeStamp, LogMessage)
  _     = span.finish()
} yield res
```

Листинг 6.3.4. *Шаблон за додавање ознака и логова*

7. ЕВАЛУАЦИЈА И РЕЗУЛТАТИ

У овом поглављу ћемо приказати различите случајеве дистрибураних трагова методе sendUpdate у чијем извршавању учествују чак четири сервиса – *proxy*, *contacts*, *messages* и *updates*.

Први случај (*слика 7.1*) приказује дистрибуирани траг уколико је метода успјешно извршена, односно, уколико су све поруке успјешно послате контактима.

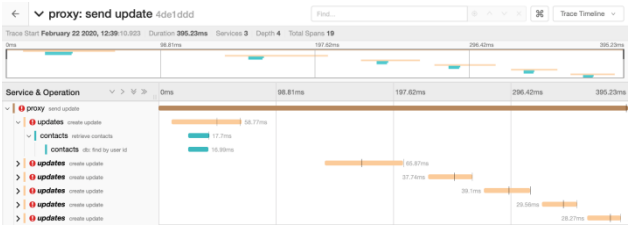


Слика 7.1. *Дистрибуирани траг успјешно извршене методе*

На *слици 7.2.* приказан је дистрибуирани траг уколико је недоступан сервис који је задужен за добављање контаката.

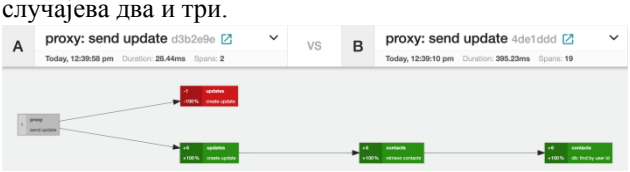
Слика 7.2. *Приказ дистрибуираног трага уколико је недоступан сервис за добављање контаката*

Трећи случај приказује дистрибуиран траг уколико је недоступан сервис задужен за слање порука. У дистрибуираном трагу приказаном на *слици 7.3*. видимо велики број индикатора неуспјешности *span*-ова који представљају сваки позив ка недоступном сервису.



Слика 7.3. *Приказ дистрибуираног трага уколико је недоступан сервис за слање порука*

Jaeger алат за анализу нам омогућава и упоређивање дистрибуираних трагова, што представља веома користан алат уколико се ради о дистрибуираним траговима са сложеним токовима (енг. *flow*). *Слика 7.4.* приказује упоредбу дистрибуираних трагова из случајева два и три.



Слика 7.4. *Упоредба трагова методе sendUpdate*

Поређењем дистрибуираних трагова видимо да се у првом приказаном трагу успјешно позвао само *updates* сервис, док су се током другог трага успјешно добавили и контакти неопходни за обраду захтјева, али је усљед отказа на сервису за слање порука дистрибуирани траг резултирао грешком.

8. ЗАКЉУЧАК

У овом раду приказан је дистрибуирани систем базиран на микросервисној архитектури који, уз коришћење *ZIO Telemetry* библиотеке, има способност прикупљања и слања дистрибуираних трагова на екстерни алат за анализу. Дате су теоријске основе технике дистрибуираног праћења и описани најзначајнији стандарди у овом тренутку. Дефинисан је појам микросервисне архитектуре и истакнути њени недостаци са аспекта надгледања система. Даље, описана је спецификација система, те приказана његова архитектура, описане *ZIO* и *ZIO Telemetry* библиотеке које чине основу саме имплементације, те детаљно објашњен процес инструментације изворног програмског кода. Такође, дате су основне одлике *Jaeger* алата, коришћеног као екстерни алат за анализу и обраду прикупљених дистрибуираних трагова. На основу анализираних дистрибуираних трагова у *поглављу 7* закључујемо да дистрибуирано праћење доноси прегршт погодности приликом надгледања система и као такво готово да га је немогуће изоставити, како из мјера предострожности и могућности отказа појединачних сервиса, тако и због појаве неправилности у самом раду појединачних компоненти. Захваљујући дистрибуираном праћењу, јасно су уочљиве операције у систему код којих се десила грешка приликом извршавања, а њихова интеграција са додатним алатима пружа чврсту подлогу за алармирање особа надлежних за одржавање доступности система.

ЛИТЕРАТУРА

[1] Benjamin H. Sigelman, Luiz André Barroso, Mike Burrows, Pat Stephenson, Manoj Plakal, Donald Beaver, Saul Jaspan, Chandan Shanbhag “Dapper, a Large-Scale Distributed Systems Tracing Infrastructure”, *Google Technical Report*, април 2010.

[2] J. Mace, R. Roelke, R. Fonseca “Pivot Tracing: Dynamic Causal Monitoring for Distributed Systems”, *SOSE* (pp. 378-393), 2015.

[3] OpenCensus, [На мрежи]. Доступно на: https://opencensus.io/ [Посљедњи приступ 5. март 2020]

[4] OpenTracing, [На мрежи]. Доступно на: https://opentracing.io/ [Посљедњи приступ 5. март 2020]

[5] OpenTelemetry, [На мрежи]. Доступно на: https://opentelemetry.io/ [Посљедњи приступ 5. март 2020]

[6] Martin Fowler - Microservices, [На мрежи]. Доступно на: https://martinfowler.com/articles/microservices.html [Посљедњи приступ 5. март 2020]

[7] Raja R. Sambasivan, Rodrigo Fonseca, Ilari Shafer, Gregory R. Ganger “So, you want to trace your distributed system? Key design insights from years of practical experience”, *CMU-PDL-14-102*, април 2014.

[8] ZIO documentation, [На мрежи]. Доступно на: https://zio.dev/docs/overview/overview_index [Посљедњи приступ 5. март 2020]

[9] Jaeger documentation, [На мрежи]. Доступно на: https://www.jaegertracing.io/docs/1.14/ [Посљедњи приступ 5. март 2020]

Кратка биографија:

Драгутин Марјановић рођен је 21. октобра 1994. године у Добоју. Године 2013, уписује Факултет техничких наука у Новом Саду, смјер *Софтверско инжењерство и информационе технологије.* Основне академске студије завршава 2017. године просјечном оцјеном 10,00 радом на тему *Имплементација софтвера за аутоматско препознавање цртежа употребом Spark алата* највишом оцјеном. Мастер академске студије, смјер *Софтверско инжењерство*, завршава 2020. године просјечном оцјеном 10,00 радом на тему *Употреба ZIO Telemetry библиотеке за прикупљање дистрибуираних трагова у системима базираним на микросервисној архитектури.*