Зборник радова Факултета техничких наука, Нови Сад

UDK: 004.9
DOI: https://doi.org/10.24867/25BE40Soic

МИГРАЦИЈА МИКРОСЕРВИСНЕ АПЛИКАЦИЈЕ СА РЕЛАЦИОНОГ МОДЕЛА ПОДАТАКА У НЕРЕЛАЦИОНИ МОДЕЛ КОРИШЋЕЊЕМ *SERVERLESS AWS* СЕРВИСА

MIGRATION FROM MICROSERVICES APPLICATION ARCHITECTURE WITH RELATIONAL DATABASES TO AWS SERVERLESS BASED SOLUTION WITH NON-RELATIONAL DATABASE

Сенка Шоић, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У раду је приказана конверзија веб апликације са микросервисном арихитектуром и релационим моделом података у апликацију нерелационог модела података која се ослања на AWS сервисе.*

Анализирана је микросервисна архитектура наспрам serverless архитектуре. Описани су коришћени AWS сервиси. Анализирани су и описани кораци миграције у serverless систем.

Кључне речи: *Микросервиси, рачунарство у облаку, Serverless, AWS, DynamoDB, AppSync, Lambda функције*

Abstract – *This paper presents conversion of web application with microservices architecture and relational data model into non-relational data model application which is relying on AWS services. It analyses microservices architecture compared to serverless architecture. It describes used AWS services. It analyses and describes migration steps into serverless system.*

Keywords: *Microservices, Cloud computing, Serverless, AWS, DynamoDB, AppSync, Lambda functions.*

1. УВОД

Микросервисна архитектура је чест избор за архитектуру система приликом развоја апликација. Разлог за то је то што се индивидуални микросервиси могу независно модификовати, испоручивати и скалирати. Као резултат, у микросервисној архитектури, развој апликације и управљање инфраструктуром су уско повезани.

Ипак, са повећаном популарношћу *serverless* парадигме и њених предности, попут изостајања потребе управљања инфраструктуром, као и модела плаћања у коме се наплаћују само искоришћени ресурси, долази до раста интереса за рефакторисањем апликација базираним на микросервисима и миграције у *serverless* архитектуру. Један од најпознатијих *cloud* провајдера – *AWS* нуди бројне сервисе који се могу искористити за изградњу *serverless* апликација.

У склопу овог рада миграција је описана на примеру једног система који поседује микросервисну архитектуру. У питању је систем за изнајмљивање аутомобила (*Rent-a-Car*), који се састоји из неколико микросервиса, корисничке (тзв. фронтенд) апликације и релационог модела података који је изграђен у *MySql* бази података.

Као резултат овог рада добијено је *serverless* решење, чији је модел података изграђен у нерелационој *NoSql* бази података (*DynamoDB*), коришћењем *AppSync, Cognito* и *Lambda* сервиса.

2. ТЕОРИЈСКЕ ОСНОВЕ

2.1. Микросервисна архитектура

Микросервисна архитектура је стил архитектуре софтверских система у ком је апликација подељена на мале компоненте које су самосталне, лабаво повезане независно испоручиве и скалабилне. Те мале компоненте се називају микросервиси и типично имају следеће карактеристике [1]:

- Имају своју комбинацију технологија (енгл. *tech stack*), библиотека и зависности
- Обично имају своју базу података
- Програмери могу да раде на њима одвојено, без утицаја на рад других тимова
- Међусобно најчешће комуницирају преко *REST API*-ја, стриминга догађаја или посредника у преносу порука (енгл. *message broker*)

Микросервиси су настали из потребе да се олакша развој и скалирање апликација када су оне постајале огромне и претешке за сналажење у бази кода, одржавање и руковање [1].

2.2 Рачунарство у облаку

Концепт рачунарства у облаку (енгл. *Cloud computing*) може се објаснити као технологија која омогућава коришћење различитих информатичких услуга на физички удаљеним серверима уз помоћ наше мрежне инфраструктуре и одговарајућих интернет протокола. *Cloud* технологија се заснива на томе да сви подаци који су неопходни кориснику (биле то апликације, документи, хардвер или нешто друго) буду доступни

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

у сваком тренутку, те *cloud* на неки начин представља испоруку услуга уместо самог производа [2].

Постоје три основна типа рачунарства у облаку [3]:

1. Инфраструктура као сервис - Типично омогућава приступ и управљање: мрежним ресурсима, рачунарима (виртуелним или на наменском хардверу) и месту за складиштење података.

2. Платформа као сервис - Уклања потребу за управљањем комплетном инфраструктуром већ омогућава фокусирање само на испоручивање и управљање самом апликацијом, јер се брига о заузимању ресурса, одржавању софтвера и сл. препушта *cloud* провајдеру.

3. Софтвер као сервис - Пружа комплетан производ који одржава провајдер. У овом случају се не мора размишљати о томе како одржавати сервис или о његовој инфраструктури, већ само о томе на који начин ће овај софтвер бити коришћен.

2.3 *Serverless computing*

Рачунарство без сервера (енгл. *serverless* computing) је *cloud* парадигма која омогућава програмерима да изграђују апликације и сервисе без потребе да обезбеђују и управљају њиховим серверима. У *serverless* архитектури комплетном позадинском инфраструктуром управља *cloud* провајдер, и једино што програмери раде је писање и испоручивање кода. *Serverless* архитектура је вођена догађајима (енгл. *event-driven*) и може бити иницирана извршавањем разних догађаја, попут акција корисника или промене података. Такође је високо скалабилна, јер *cloud* провајдер може аутоматски покренути нове инстанце кода као одговор на повећан саобраћај [4].

2.4 Микросервиси или *Serverless*, коју архитектуру одабрати?

Овде нема једног тачног одговора. Све зависи од случаја коришћења као и неких субјективних преференци.

Serverless архитектура је добар избор ако нам је потребна брза апликација вођена догађајима. Такође, ако нам је потребно да брзо дођемо до радне верзије апликације, без велике потребе или жеље за управљањем инфраструктуром, онда је ово добар приступ, јер омогућава брзу изградњу лаганих, флексибилних апликација које могу бити проширене или ажуриране релативно брзо.

Међутим, за комплексније и дуготрајније процесирање података, микросервиси су вероватно прихватљивије решење, како се инфраструктура и *tech stack* могу адаптирати по потреби [4].

2.5 Амазон веб сервиси

Амазон веб сервиси (енгл. *Amazon Web Services – AWS*) су обимна, стално развијајућа *cloud* платформа коју пружа Амазон. Укључује мешавину инфраструктура као сервис, платформа као сервис и софтвер као сервис понуда. *AWS* сервиси нуде организационе алате, попут рачунарске снаге, складишта база података и сервиса за испоручивање садржаја [5].

2.6 *DynamoDB*

Amazon *DynamoDB* је потпуно управљана, нерелациона (енгл. *noSQL*) база података која омогућава брзе и предвидиве перформансе и конзистентну скалабилност. Основне компоненте [6]:

Табела - *DynamoDB* складишти податке у табелама. Табела је колекција података (ставки).

Ставка - Група атрибута која се може јединствено идентификовати унутар ставки једне табеле, пандан редовима у неким релационим системима база података.

Атрибут - Свака ставка се састоји од једног или више атрибута. Атрибут је фундаментални елемент података, нешто што нема потребе да се даље растављa. Атрибути у *DynamoDB* табели се могу поредити са колонама у релационим системима база података, са разликом да се осим примарног кључа унапред не дефинишу називи и типови атрибута

Примарни кључ – Јединствено идентификује сваку ставку табеле, тако да ниједна ставка не може имати исти примарни кључ. Постоје два различита типа примарних кључева:

1. Партициони кључ - Једноставни примарни кључ, састоји се од једног атрибута познатог као партициони кључ

2. Композитни примарни кључ - Састоји се од два атрибута: партициони кључ и кључ за сортирање (тзв. сорт кључ)

У табели која има партициони кључ и кључ за сортирање могуће је да више ставки има исту вредност партиционог кључа, али такве ставке морају имати јединствену вредност сорт кључа, како би могле бити јединствено идентификоване. Све ставке са истом вредношћу партиционог кључа су складиштене заједно, у истој партицији, сортиране по вредности сорт кључа.

Секундарни индекс - Секундарни индекс нам омогућава да вршимо упите над базом користећи алтернативни кључ, поред коришћења примарног кључа. Постоје:

1. Глобални секундарни индекси (партициони кључ и сорт кључ се могу разликовати од оних у табели)

2. Локални секундарни индекси (исти партициони кључ, али различит сорт кључ)

DynamoDB Streams – опционо својство „хватања“ догађаја модификације података у *DynamoDB* табелама. Подаци о овим догађајима се појављују у реду у приближно реалном времену, у редоследу у

ком су се догађаји десили. Сваки догађај је представљен записом (енгл. *stream record*) и он се креира приликом догађаја додавања, ажурирања или брисања ставке из табеле.

Могуће је користити *DynamoDB Streams* заједно са *AWS Lambda* функцијама, да се креирају окидачи (енгл. *trigger*) – код који се извршава аутоматски када год се догађај од интереса појави у *stream*-у, што је и искоришћено у овом раду.

2.7 *GraphQL*

GraphQL је језик за креирање упита и окружење за извршавање тих упита над постојећим подацима, користећи унапред дефинисану шему типова података. Обезбеђује комплетан и разумљив опис података, даје клијентској страни моћ да тражи и добије тачно оно што јој је потребно од података, и омогућава добављање више ресурса у једном захтеву праћењем референци између њих. Главни концепти су: **шема** (сет типова који у попуности описује сет података над којим је могуће извршавати упите), **тип** (тип објекта који је могуће добавити), **упит** (служи за читање података), **мутација** (служи за ажурирање података) [7].

2.8 *AppSync*

AWS AppSync је потпуно управљан *GraphQL API* сервис који, у правом *serverless* облику, омогућава креирање *API*-ја без бриге о томе како га хостовати [8].

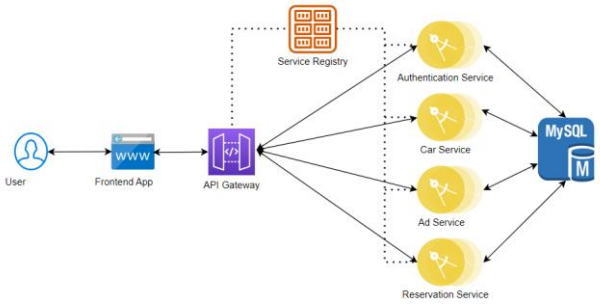
3. АРХИТЕКТУРА МИКРОСЕРВИСНОГ СИСТЕМА

Rent-a-Car систем се састоји од корисничке (фронтенд) апликације која комуницира са микросервисима преко *Zuul API Gateway*-а (Слика 1) Систем је направљен тако да подржи кључне функционалности:

- Регистрација и пријављивање корисника
- Креирање аутомобила који ће бити повезан са одређеним корисником и за који ће он моћи да креира огласе за изнајмљивање
- Креирање огласа за изнајмљивање аутомобила
- Претраживање и филтрирање доступних огласа
- Изнајмљивање огласа

На основу ових функционалности и главних ентитета у систему произашла су четири микросервиса:

- Аутентификациони микросервис – чува податке о корисницима и задужен је за ауторизацију пристиглих захтева
- Сервис аутомобила – чува информације о аутомобилима у систему
- Сервис огласа – чува податке о огласима у систему
- Сервис резервација – бави се чувањем креираних захтева за резервације и самих резервација



Слика 1 - Дијаграм архитектуре Rent-a-Car апликације

4. ИМПЛЕМЕНТАЦИЈА

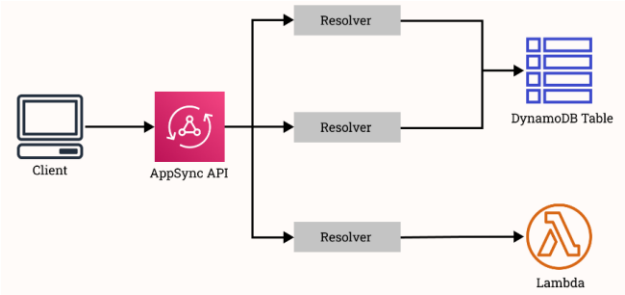
4.1 Миграција базе из релационе у нерелациону

Применом познатих концепата и стратегија моделовања *DynamoDB* табеле, а полазећи од потребних шаблона приступа које апликација треба да задовољи, добијен је следећи модел:

Главна табела апликације је табела чији партициони кључ представља *ID* корисника, сорт кључ представља *ID* Огласа, Аутомобила или Резервације, а сваки *ID* ће започињати префиксом *AD#, CAR#* или *RES#* како би се приликом претраживања могла разликовати ова три ентитета. На овај начин сви огласи, аутомобили и резервације једног корисника су груписани у истој партицији, што је врло корисно, узимајући у обзир карактеристичне шаблоне приступа које треба задовољити, а који су везани за добављање поменутих података за једног корисника.
Веза између Огласа и Аутомобила је представљена дуплирањем података за Аутомобил у сваком Огласу, како би им се могло директно приступити у сваком огласу, без додатних референцирања или упита, а из разлога што је учесталост измене података Аутомобила занемарљива. Што се тиче Огласа и Резервација, за њихову везу је могуће искористити глобални секундарни индекс. Наиме, у ставку Резервације је само потребно додати атрибут *ID* Огласа, који ће се искористити као партициони кључ глобалног секундарног индекса, а сорт кључ остаје исти као у главној табели.
Да би се подржало претраживање Огласа по локацији, направљен је још један глобални индекс чији ће партициони кључ бити локација огласа, а сорт кључ остаје исти.

4.2 Коришћење *AppSync*-а

Да би се брзо и једноставно могао направити *API* који подржава све основне операције над *DynamoDB* моделом, искоришћен је *AppSync* сервис. Уз помоћ њега креирана је *GraphQL* шема података која је потом повезана са *DynamoDB* табелом која представља њен извор података, а да би се функционалност комплетирала креирани су потребни резолвери *GraphQL* модела, који представљају спону између *GraphQL* захтева и одговора са *DynamoDB* табелом.



Слика 2 –Гграфички приказ AppSync комуникације [8]

4.3 Коришћење *Lambda* функције и *DynamoDB Stream*-а

Остаје још једна функционалност која није покривена, а то је само резервисање огласа. У *Rent-a-Car* систему је постојала улога администратора, који је ручно одобравао или одбијао пристигле захтеве, како би се избегли проблеми са конкурентношћу.
Овде је тај задатак аутоматизован коришћењем *Lambda* функције и *DynamoDB Stream*-а. Наиме, пошто је могуће „хватати“ догађаје у *DynamoDB* табели, уз помоћ *DynamoDB Stream*-а над табелом и *Lambda* функције коју иницирају догађаји у *stream*-у, то је искоришћено да би омогућили аутоматско одобравање или одбијање резервације на основу тренутне доступности огласа. Дакле, када корисник направи захтев за резервацију, иницира се извршавање *Lambda* функције која проверава доступност огласа и на основу тог резултата одобрава или одбија захтев за резервацију.

5. ЗАКЉУЧАК

Описаном миграцијом микросервисне апликације у овом раду извршена је трансформација у *serverless* апликацију са нерелационим моделом података. Добијена је апликација која има једноставни нерелациони модел који је ефективно смештен у једну табелу, уз два глобална секундарна индекса. Тиме су искоришћене бројне погодности које нуди *DynamoDB*, у виду брзих и предвидивих перформанси као и високе скалабилности.
На миграцију микросервисне архитектуре на *serverless* се одлучује све више компанија, услед бројних предности које *serverless* архитектура доноси, али пре него што се одлучи на такав корак потребно је добро размислити о томе шта она носи са собом. Сам процес је доста сложен и умногоме зависи од тренутне архитектуре, циљева и ограничења система. Познавање пословних процеса и тачно уочавање шаблона коришћења које је потребно подржати, те успешно моделовање базе података је кључна тачка успешне миграције.
Лоше постављен модел, који касније захтева промене може донети више штете него користи. Зато је први корак моделовања података уствари и најважнији корак миграције, и потребно му је поклонити највише пажње. Даље је веома важно идентификовати компоненте и функционалности које могу бити мигриране у *serverless* функције или сервисе, као и рефакторисати код и податке да би се уклопили и прилагодили serverless моделу.

6. ЛИТЕРАТУРА

[1] What are Microservices? https://cmcglobal.com.vn/cloud-migration/what-are-microservices-microservices-in-cloud-computing-explained

[2] Рачунарство у облацима – *Cloud computing* https://www.industrija.rs/vesti/clanak/racunarstvo-u-oblacima-cloud-computing

[3] *Types of cloud computing* https://aws.amazon.com/what-is-cloud-computing

[4] *Serverless vs Microservices – Which Architecture to Choose?* https://www.solutelabs.com/blog/serverless-vs-microservices

[5] *Amazon Web Services (AWS)* https://www.techtarget.com/searchaws/definition/Amazon-Web-Services

[6] *What is Amazon DynamoDB?* https://docs.aws.amazon.com/amazondynamodb/latest/developerguide/Introduction.html

[7] https://graphql.org/

[8] *What is AWS AppSync* https://sst.dev/chapters/what-is-aws-appsync.html

Кратка биографија:

Сенка Шоић рођена је 1997. године у Врбасу. Завршила је основне академске студије 2020. године на Факултету техничких наука у Новом Саду. Уписала је мастер студије исте године, студијски програм Рачунарсво и аутоматика - Електронско пословање.