Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/10BE25Kalabic

УПОРЕДНА АНАЛИЗА РАЗВОЈА SERVERLESS И CONTAINERIZED АПЛИКАЦИЈА НА AMAZON ПЛАТФОРМИ

COMPARATIVE ANALYSIS OF SERVERLESS AND CONTAINERIZED APPLICATION DEVELOPMENT ON AMAZON PLATFORM

Јелена Калабић, *Факултет техничких наука, Нови Сад*

Област – РАЧУНАРСТВО И АУТОМАТИКА

Кратак садржај – *У раду је пружена упоредна анализа развоја апликација са различитим архитектурама (serverless и containerized-microservice) на Amazon платформи. Такође, дат је преглед релевантних концепата и опис архитектура две апликације.*

Кључне речи: *Serverless, containerized, микросервис, AWS сервиси, Amazon*

Abstract – *This paper provides a comparative analysis of serverless and containerized-microservice application development on Amazon platform. An overview of relevant concepts and a description of two application architectures are given.*

Keywords: *Serverless, containerized, microservice, AWS services, Amazon*

1. УВОД

Архитектура софтвера одувек је била изузетно важна за развој софтвера и представља један од фактора који у значајној мери утичу на успех софтверског производа на тржишту. Током времена разни типови архитектура су долазили до изражаја и уживали велику популарност (попут монолитне, микросервис и *serverless* архитектуре).

У последњој деценији посебну популарност доживели су концепти попут контејнера (*container*) и *cloud computing*-а и чини се да је без њих тешко замислити савремени развој софтвера. Најпознатији и највећи *cloud* провајдери на тржишту тренутно су: *Amazon Web Services* (AWS), *Microsoft Azure* и *Google Cloud*. Могућности су велике и посебан је изазов одлучити се које технологије и који приступ су најпогоднији за развој одређеног производа.

Да ли развијати апликацију независну од *cloud* провајдера која може да се извршава практично било где? Или у потпуности користити сервисе које *cloud* провајдери нуде и фокусирати се само на развој пословне логике?

У наставку рада дат је преглед релевантних концепата, приказане су архитектуре две апликације и дато је поређење *containerized-microservice* и *serverless* приступа.

НАПОМЕНА:
Овај рад проистекао је из мастер рада чији ментор је био др Бранко Милосављевић, ред. проф.

2. ПРЕГЛЕД РЕЛЕВАНТНИХ КОНЦЕПАТА

Претходних десетак година обележила је појава *cloud computing*-а који је значајно променио развој и одржавање софтверских производа. Шта је заправо ***cloud computing***? „*Cloud computing* је испорука захтеваних ИТ ресурса путем интернета. Модел плаћања заснива се на томе да се ресурси плаћају само онда када су заиста у употреби. Уместо да купујете, поседујете и одржавате физичке центре података (*data center*) и сервере, можете по потреби приступати технолошким услугама, попут рачунарске снаге (*computing power*), складиштења података (*storage*) и база података (*database*), које пружају *cloud* провајдери попут AWS-а“ [1]. Један од највећих играча на тржишту *cloud computing*-а јесте **AWS**. „AWS представља платформу веб сервиса који нуде решења за рачунарску снагу, складиштење и умрежавање на различитим нивоима апстракције. На пример, складиштење података може бити на нивоу блока (*block-level storage)* или високо дистрибуирано складиште објеката (*distributed object storage*)“ [2].

Платни модел заснива се на принципу плаћања по употреби (*pay-per-use*). Са појавом *cloud*-а поједини типови софтверских архитектура дошли су до изражаја, а у наставку ће бити описана два типа која су значајна за тему овог рада - serverless архитектура и контејнерска микросервис (*containerized-microservice*) архитектура. ***Serverless* архитектуру** тешко је дефинисати, али оно око чега се сви слажу јесте да *serverless* не значи да не постоје сервери на којима се извршава код већ да програмери/корисници не морају да брину о њима (не морају да покрећу и одржавају виртуелне или физичке сервере). *Serverless* пружа прилику програмерима да се фокусирају на писање кода уместо на одржавање сервера и база података.

У *serverless* свету *cloud* провајдери попут AWS-а динамички управљају доделом и дистрибуцијом ресурса. **Микросервиси** (*microservice*) представљају архитектонски приступ развоју софтвера заснован на изради апликације као колекције малих, независних сервиса.

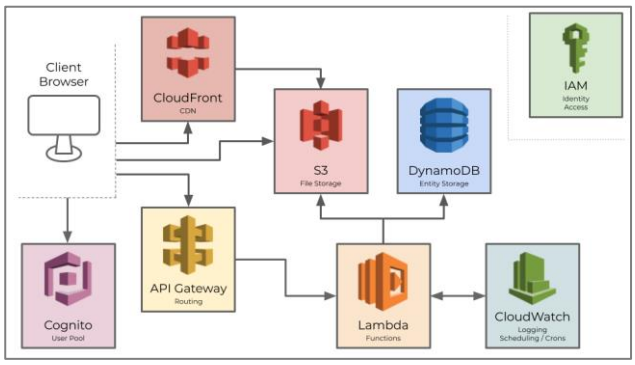
Са појавом и развојем *cloud*-а, тачније са еволуцијом контејнерских (*container*) архитектура појавио се и концепт изворних *cloud* микросервиса (*cloud native microservices*). „**Контејнер** представља старндардну јединицу софтвера која пакује код и све потребне зависности (*dependencies*) што омогућава брзо и

поуздано покретање апликације у различитим рачунарским окружењима“ [3]. Микросервиси се често испоручују у једном или више контејнера.

С обзиром да су контејнери изолована окружења, могу се користити за *deploy*-овање микросервиса брзо и безбедно без обзира на програмски језик који се користи за развој микросервиса.

3. ОПИС АРХИТЕКТУРА АПЛИКАЦИЈА

На слици 1. приказана је *serverless* архитектура апликације за складиштење и управљање корисничким фотографијама.



Слика 1. *Serverless архитектура апликације*

На слици 1. види се **IAM** компонента неопходна да би један сервис приступао другом и да би се дефинисала одређена права приступа те је она имплицитно повезана са свим компонентама система и те везе су изостављене зарад једноставности саме слике.

Клијентска апликација (***Client***) представља SPA (*Single Page Application*) апликацију развијену у *JavaScript* програмском језику уз употребу *Vue.js framework*-а. Клијентска апликација ускладиштена (*hosted*) је у **S3** складишту.

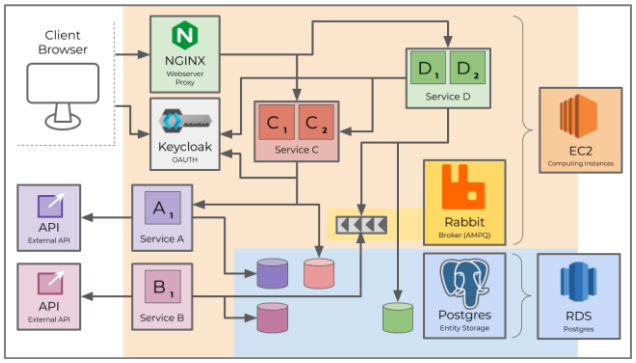
Статички садржај клијентске апликације испоручује се корисницима путем ***CloudFront***-а. *CloudFront*-ова главна улога је брже испоручивање садржаја крајњим корисницима. S3 се осим за складиштење клијентске апликације користи и за складиштење корисничких фотографија и њихових умањених верзија.

За регистровање, аутентификацију и ауторизацију корисника задужена је ***Cognito*** компонента. Сви захтеви ка серверском делу апликације рутирају се кроз API *Gateway* компоненту што резултује активирањем одређене ламбда функције (***Lambda***). Ламбда функције имплементиране су у *framework*-у Node.js-а – *Express*-у.

DynamoDB представља базу података у којој се складиште подаци о фотографијама и албумима.

Сви записи (*logs*) о активности ламбда функције и евентуалним проблемима који се могу јавити завршавају у компоненти ***CloudWatch***. Слика 2. приказује *containerized-microservice* архитектуру апликације која олакшава свакодневни рад запосленима у компанији која се бави дигиталним оглашавањем/рекламирањем (*digital advertising*).

На слици 2. могуће је уочити микросервисе, њихове међусобне везе као и везе ка екстерним платформама. Клијентска апликација (***Client***)



Слика 2. *Containerized-microservice архитектура апликације*

развијена је у *JavaScript* програмском језику уз употребу *Vue.js framework*-а. Као што је приказано на слици, клијентска апликација има две директне везе – са ***Keycloak*** сервисом и **NGINX** веб сервером. *Keycloak* сервис задужен је за аутентификацију корисника приликом пријављивања у систем. **NGINX** веб сервер задужен је за испоручивање клијентске апликације и за прослеђивање (*proxying*) свих захтева ка сервисима. NGINX не приступа свим сервисима из безбедносних разлога.

Сви сервиси који су обухваћени розе бојом у продукционом окружењу извршавају се унутар контејнера, при чему сервиси могу имати једну (нпр. сервис A) или више покренутих инстанци (нпр. сервис D). Сви контејнери, унутар којих се извршавају микросервиси, покренути су унутар AWS **EC2** инстанци. Као систем за управљање базама података користи се *Amazon* **RDS**. Сваки микросервис има сопствену базу података, али уколико се сервис извршава у више контејнера они сви деле исту базу података. Већина микросервиса имплементирана је у *Java* програмском језику уз употребу *Spring Boot framework*-а док су поједини сервиси имплементирани *JavaScript* програском језику уз употребу *Express framework*-а. За контејнере се користи *Docker*, брокер за размену порука је *RabbitMQ* а *PostgreSQL* користи се као база података.

4. ПОРЕЂЕЊЕ АРХИТЕКТУРА

У претходном поглављу описане су две апликације, са две различите архитектуре и оне су предмет поређења у овом поглављу.

4.1. Сложеност развоја

За поређење сложеност развоја изабрани су следећи параметри поређења: време и напор потребни да би се савладала/научила употреба технологија и сервиса неопходних за развој апликације, слобода при избору технологија, контрола над сервером и извршним окружењем, конфигурација сервиса, време потребно за *deployment*, комплексност тестирања и *debug*-овања, могућност креирања локалног (*offline*) извршног окружења.

За *serverless* приступ приликом развоја апликације морамо бити спремни да уложимо време и труд за изучавање *cloud* сервиса које желимо да користимо да бисмо били у стању да их исправно конфигуришемо и омогућимо да међусобно комуницирају као и да научимо да користимо *cloud* алате и корисничке интерфејсе. Додатнуману представља чињеница да стечено знање углавном није применљиво приликом употребе сервиса другог *cloud* провајдера.

Уколико се одлучимо за развој апликације са контејнерском микросервис архитектуром, највероватније ћемо се одлучити за технологије и сервисе које већ довољно добро познајемо што значи да је потреба за усвајањем нових знања мања него код *serverless* приступа. Када је реч о слободи приликом избора технологије, ту је дефинитивно предност на страни контејнерског микросервис приступа јер можемо да изаберемо било коју технологију/-програмски језик. *Serverless* приступ са собом носи одређена ограничења приликом избора технологије јер *cloud* провајдери подржавају ограничен скуп технологија.

Серверска страна *cloud* сервиса тј. комплетна инфраструктура на којој се заснива апликација са *serverless* архитектуром и окружење у коме се извршава су под контролом *cloud* провајдера и оно што сам корисник може да конфигурише је прилично ограничено.

С друге стране, употребом контејнера ми смо ти који су у потпуности задужени за избор оперативног система, библиотека, технологија, за бригу о потребним ресурсима, за дефинисање конфигурационих фајлова (попут *Dockerfile*-а) и слично. Међутим, када је реч о комплексности конфигурације сервиса предност је на страни *serverless* приступа.

Када је реч о времену потребном за *deployment* апликације, иако се може рећи да су оба приступа мање-више слична, блага предност ипак се може дати *serverless* приступу (јер контејнерима иницијално треба више времена за конфигурисање системских подешавања, библиотка и слично).

Апликације са *serverless* архитектуром тешко је тестирати јер је серверско окружење тешко реплицирати у локалном окружењу. С друге стране, контејнери се извршавају исто без обзира у ком окружењу (продукционом, тест) су покренути. И када је у питању *debug*-овање апликације предност је на страни контејнерске микросервис архитектуре.

Оно што може бити од значаја за програмере при изршавању њихових задатака приликом развоја апликације јесте могућност покретања свих сервиса неопходних за рад целе апликације локално (идеално без потребе да имају интернет конекцију - *offline*).

Чињеница да *serverless* приступ у потпуности почива на употреби *cloud* сервиса чини практично немогућим да се апликација у потпуности покрене локално.

С друге стране, употребом контејнера могуће је локално покренути све неопходне сервисе.

4.2. Сложеност инсталације и одржавања

Пре самог поређења апликације са *serverless* архитектуром и апликације са контејнерском микросервис архитектуром која се извршава на „голом“ (*bare*) серверу са аспекта сложености инсталације и одржавања, побројан је део активности које су обухваћене процесима инсталације и одржавања даби се стекао утисак о њиховој комплексности.

Активности које је обавезно спровести су следеће: активности у вези са постојећим захевима када је реч о саобраћају и доступности, подешавање мрежне инфраструктуре, подешавања рачунарских ресурса, прављење резервних копија података, дефинисање метрика и упозорења на нивоу система и платформе, писање неопходне документације и слично.

У склопу одржавања неопходно је редовно проверавати да ли је потребно ажурирати рачунарске ресурсе, оперативни систем, системске сервисе и слично, тестирати нове верзије, мењати конфигурације уколико је потребно, надгледати параметре попут оптерећења процесора, заузетости меморије, саобраћаја, тестирати могућност обнове података на основу резервних копија и слично.

Једна од основних предности *serverless* приступа јесте чињеница да је *cloud* провајдер задужен за спровођење свих (или бар већине) ових активности. С друге стране, у случају контејнерског микросервис приступа сва одговорност је на људима који се баве развојем и одржавањем апликације.

4.3. Трошкови продукције

Cloud провајдери пружају калкулаторе који корисницима омогућавају да израчунају потенцијалне трошкове употребе *cloud* сервиса. Са друге стране, када одлучимо да изнајмљујемо сервере имамо информацију о цени сервера коју одређују карактеристике сервера попут меморије, CPU-а и SSD диска. Из тог разлога у наставку се приказују конкретни трошкови само за рачунарске ресурсе (*compute*). Наплаћивање AWS сервиса функционише по принципу да се плаћа онолико колико се заправо троши. У случају **ламбда** функција то значи да се плаћа у складу са бројем захтева и временом извршавања функције као и потребном меморијом. На месечном нивоу бесплатно је првих милион захтева након чега се плаћа 0,2 $ за милион захтева. Када је реч о цени извршавања функција параметри који утичу на цену су време и меморија потребни да би се функција извршила. На месечном нивоу бесплатно је 400.000 GB-s а након тога се плаћа 0,00001667 $ по GB-s. С друге стране, када је реч о ценама за изнајмљивање сервера на *DigitalOcean*-у оне зависе од карактеристика саме машине (меморија, CPU, SSD диск) и цене из стандардне понуде крећу се од 5 до 960 $ месечно. **У првом случају** ћемо претпоставити да имамо апликацију која се често користи али не захтева пуно меморије и времена да би се извршавала. Рецимо да имамо следеће карактеристике: број захтева – 40.000.000, време потребно за извршавање – 0,3 s а потребна меморија 128 MB.

Резултати прорачуна су показали да би трошкови за AWS *Lambda* сервис били 26,137 $ а за DigitalOcean сервере са карактеристикама 1 GB меморије, 1 CPU, 25 GB SSD – минимални трошкови били би 50 $. Трошкови за ламбду су скоро па дупло мањи те се може закључити да је у овом случају ламбда бољи избор. **У другом случају** ћемо претпоставити да имамо апликацију која се користи за неке захтевне обраде или анализе велике количине података што значи да је временски и меморијски захтевна а да је интензитет саобраћаја мањи.

Рецимо да имамо следеће карактеристике: број захтева – 1.000.000, време потребно за извршавање – 2 min тј. 120 s а потребна меморија 3008 MB. Резултати прорачуна су показали да би трошкови за AWS *Lambda* сервис били 5869,507 $ а за DigitalOcean сервере са карактеристикама 3 GB меморије, 1 CPU, 60 GB SSD – минимални трошкови били би 1410 $. Из ових резултата може се закључити да су трошкови за *DigitalOcean* сервере мањи. На примеру *serverless* апликације приказане у овом раду може се уочити да се поред ламбда функција користе и други AWS сервиси (S3, *DynamoDB*, *Cognito*, API *Gateway*, *CloudFront*). Уколико бисмо желели да идентичну апликацију развијемо и извршавамо на серверима *DigitalOcean*-а морали бисмо пронаћи open-source (јавно доступне) алтернативе (попут *MinIO*, *MongoDB*, *Keycloak*, *nginx*, *CloudFlare*), инсталирати их и конфигурисати у сладу са нашим потребама. То би последично само повећало број сервера неопходних за извршавање апликације а самим тим би и трошкови порасли.

4.4. Скалабилност

Када је реч о скалабилности ламбда функције може се рећи да је то једна од њених основних предности и карактеристика јер AWS аутоматски скалира ламбда функцију. С обзиром да је код функције *stateless* AWS ламбда може да покрене онолико копија функције колико је потребно. AWS ламбда ће динамички алоцирати капацитет у складу са долазним саобраћајем – када има пуно захтева које треба обрадити биће покренуто више инстанци, у супротном када нема пуно захтева стопираће претходно покренуте инстанце. Број инстанци које се могу извршавати у паралели зависи од региона и креће се од 500 до 3.000. Оно што карактерише све AWS сервисе па самим тим и ламбду јесте да су високо доступни (*highly available*).

С друге стране, када је реч о скалабилности сервера ту је прича много другачија. Ми смо ти који су дужни да обезбеде довољан број сервера да би се успешно обрадили сви долазни захтеви. То подразумева свакодневно надгледање (*monitoring*) употребе ресурса сервера и да ли је тренутна инфраструктура у стању да се избори са долазним саобраћајем. Када се уочи потреба за новим серверима, потребно их је изнајмити, затим обавити сву потребну конфигурацију и инсталацију неопходну за извршавање апликације. Такође, за доступност сервиса смо ми задужени што значи да све потенцијалне проблеме који доводе до прекида нормалног извршавања аликације морамо бити у стању што пре да решимо јер време када апликација није доступна носи за последицу новчане губитке.

5. ЗАКЉУЧАК

Основна идеја рада била је да се прикажу предности и мане два различита приступа развоја софтверског производа тј. предности и мане развоја апликација са *serverless* и контејнерском микросервис архитектуром. Конкретан одговор на питање који приступ је бољи није могуће дати јер избор приступа у великој мери зависи од домена, случајева употребе (*use cases*), ограничавајућих спољашњих фактора попут расположивог буџета, људских ресурса, временских рокова и слично. Међутим, чини се да је *serverless* приступ погодан онда када је код једноставан тј. када пословна логика није комплексна са аспекта ресурса и времена извршавања, када се апликација употребљава периодично или једноставно није могуће предвидети саобраћај а желимо скалабилно решење.

С друге стране, када имамо ресурсно захтевне и дуготрајне обраде или када имамо комплексну апликацију са уједначеним саобраћајем (у смислу да је сервер већи део дана заузет обрађивањем захтева) онда је контејнерски микросервис приступ боља опција од *serverless* приступа. Када је реч о идејама за наставак истраживања на ову тему, било би занимљиво приказати и истражити потенцијалне алтернативе AWS сервиса код других *cloud* провајдера и направити поређење трошкова. Такође, могло би се направити поређење трошкова изнајмљивања сервера за различите провајдере као што је то урађено на примеру *DigitalOcean*-а.

6. ЛИТЕРАТУРА

[1] *What is could computing?*, https://aws.amazon.com/what-is-cloud-computing/ (приступљено у септембру 2020.)

[2] Michael Wittig, Andreas Wittig. *Amazon Web Services in Action, Second Edition*. Manning Publications Co, New York, 2018. ISBN 9781617295119

[3] *What is a Container?*, https://www.docker.com/resources/what-container (приступљено у септембру 2020.)

Кратка биографија:

Јелена Калабић рођена је 22.10.1994 године у Ужицу, Република Србија. Године 2013. завршава општи смер гимназије „Светозар Марковић“ у Новом Саду и уписује се на Факултет техничких наука, одсек Електротехника и рачунарство, смер Рачунарство и аутоматика. Године 2017. уписује мастер академске студије на истом смеру. Добитница је „Доситеја“ стипендије за постигнут успех на основним академским студијама. Године 2018. добија стипендију немачке привреде „др Зоран Ђинђић“ и шест месеци проводи на стручној пракси у Хамбургу у компанији *Esome advertising technologies*.