Зборник радова Факултета техничких наука, Нови Сад

УДК: 4.4
ДОИ: https://doi.org/10.24867/35BE17Glisic

Миграција наспрам развоја *Cloud-Native* апликација

Migration versus Cloud-Native application development

Цвијетин Глишић, *Факултет техничких наука, Нови Сад*

Студијски програм – ПРИМЕЊЕНО СОФТВЕРСКО ИНЖЕЊЕРСТВО

Кратак садржај – *У овом раду обрађене су основне карактеристике cloud computing-*а*, као и начини и приступи миграције и развоја апликација у cloud окружењу. Посебан фокус рада је на два приступа cloud-у: „replatform" миграцији, која подразумева подизање постојеће (legacy) апликације у cloud уз минималне измене ради искоришћења одређених погодности cloud платформе и cloud-native приступу, који обухвата развој нове апликације посебно намењене за рад у cloud окружењу и коришћење свих предности које cloud омогућава.*

Кључне речи: *cloud, cloud-computing, миграција, cloud native, replatform*

Abstract – *This paper explores the fundamental characteristics of cloud computing, as well as the various approaches to migrating and developing applications in cloud environments. The primary focus is on two key cloud strategies: the "replatform" migration, which involves deploying a legacy application to the cloud with minimal modifications to leverage selected cloud benefits, and the cloud-native approach, which entails building a new application specifically designed to operate in the cloud and to fully utilize all advantages offered by cloud platforms.*

Keywords: *cloud, cloud-computing, migration, cloud native, replatform*

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

1. УВОД

У савременом добу, *cloud-computing* је постао битан за развој и имплементацију апликација, омогућавајући организацијама већу флексибилност, скалабилност и ефикасност. Овај рад истражује два приступа у усвајању *cloud* технологија: миграцију постојећих апликација на *cloud*, без измене архитектуре и логике саме апликације познату као „*replatform*", и развој *cloud-native* апликација, дизајнираних да у потпуности искористе предности cloud окружења. Док „*lift and shift*" приступ омогућава брзу транзицију уз минималне измене постојеће инфраструктуре, *cloud-native* развој захтева већа почетна улагања, али нуди супериорну еластичност, отпорност и ефикасност у дугорочној перспективи [1].

Кроз упоредну анализу ових приступа, рад има за циљ да прикаже њихове предности, изазове и применљивост у различитим контекстима, пружајући смернице за доношење што бољих одлука у процесу преласка на *cloud*. За конкретан експеримент имплементирана је *Talent Acquisition Manager* апликација. Ова апликација, због своје природе и захтева за скалабилношћу, представља релевантан случај за анализу перформанси и прилагодљивости у *cloud* окружењу, и на њеном примеру су извучени закључци у контексту теме овог рада.

2. *CLOUD-NATIVE* ПРИСТУП

Cloud-native представља архитектурални стил развоја софтвера, где су апликације од темеља пројектоване и оптимизоване за рад у *cloud* окружењу. Овај приступ карактерише коришћење технологија које омогућавају максимално искоришћавање предности *cloud* платформи, попут скалабилности, еластичности и отпорности на грешке. *Cloud-native* апликације се обично заснивају на микро-сервисној архитектури, где је апликација подељена на мање, независне компоненте које међусобно комуницирају путем дефинисаних интерфејса. Оваква структура омогућава брзо прилагођавање променама, коришћење различитих, међусобно некомпатибилних технологија у истом систему, лакше скалирање појединачних делова апликације и ефикасније коришћење ресурса.

Кључне карактеристике cloud-native приступа укључују коришћење контејнера (нпр. *Docker*), оркестрацију контејнера (нпр. *Kubernetes*), аутоматизовано управљање инфраструктуром путем алата попут *CI/CD* (*Continuous Integration/Continuous Deployment*). Ове технологије омогућавају апликацијама да буду високо доступне, отпорне на кварове и способне да се аутоматски прилагођавају променама у оптерећењу. *Cloud-native* апликација може аутоматски скалирати своје ресурсе у зависности од броја корисника, смањујући трошкове током периода мање активности.

Међутим, *cloud-native* развој доноси и бројне изазове. Захтева велика почетна улагања у погледу времена и финансија. Поред тога, прелазак на микро-сервисну архитектуру може донети и додатну сложеност у управљању апликацијама, као на пример потребу за напредним мониторингом и координацијом између

сервиса. Упркос томе, *cloud-native* приступ је веома чест код нових апликација које захтевају висок степен скалабилности и флексибилности јер омогућава организацијама да у потпуности искористе сав потенцијал *cloud-computing*-а [2].

3. МИГРАЦИЈЕ НА *CLOUD*

Миграција постојећих апликација на *cloud* представља процес премештања постојећих софтверских решења са локалне инфраструктуре на *cloud* платформу уз минималне или никакве измене у њиховој основној архитектури. Овај приступ је посебно привлачан за организације које желе брзо искористити предности *cloud*-а, попут смањења трошкова за физичку инфраструктуру и побољшања доступности, без потребе за значајним преобликовањем апликација. Седам највише примењиваних и најпознатијих типова миграције су:

1. *Rehost (Lift and Shift)* – Померање апликације на *cloud* по „*as is*“ принципу, без икаквих архитектуралних промена и додатних компоненти.
2. *Relocate* – Погодно за апликације које су већ у виртуелним окружењима, попут *docker* контејнера или *kubernetes* кластера. Принцип је исти као код *rehost* приступа, али се мигрира цело окружење.
3. *Replatform* – Апликације се делимично адаптирају *cloud* окружењу, али без промене логике и архитектуре саме апликације. Ово може да подразумева прелазак на *cloud* базу података, имплементацију *HPA* (*horizontal pod autoscaling*) на нивоу целе апликације, имплементацију сервиса за праћење метрика, итд.
4. *Refactor/rearchitect* – Подразумева редизајнирање апликације, уноси видљиве промене у архитектури а често и у логици апликације. Ово је тип миграције који организације користе како би постојећу апликацију што више приближиле cloud-native стилу.
5. *Repurchase (Drop and Shop)* – Представља замену постојеће (*legacy*) апликације неким *SaaS* (*Software-as-a-Service*) решењима. Овај приступ је поготову изводљив код апликација које обрађују неке стандардизоване процесе.
6. *Retire* – Одбацивање застарелих апликација или делова апликација, ради усмеравања ресурса на битније процесе и делове система.
7. *Retain/revisit* – Задржавање одређених апликација или делова система у локалном (*on-premise*) окружењу, обично због безбедносних, регулаторних или техничких ограничења, уз план да се њихова миграција поново размотри у будућности [3][4].

4. *TALENT ACQUISITION MANAGER* АПЛИКАЦИЈА

4.1. Идеја и инспирација

С обзиром да у последње време све више људи проналази запослење путем интернета, те чињенице да се појављује све више апликација које помажу људима да пронађу посао на овај начин и повежу се с људима из исте или сличне струке, створена је идеја да софтверско решење уз овај рад буде апликација тог типа.

4.2. Коришћене технологије

У реализацији софтверског решења за овај рад коришћене су технологије и алати који су често заступљени у раду са cloud окружењем:

- *ASP.NET Core* – радно окружење отвореног типа развијено од стране компаније *Microsoft*, за развој *web* и *cloud* апликација.
- *gCloud* – cloud платформа развијена од стране компаније *Google*. Укључује рачунарске ресурсе, складиштење података, аналитiku и машинско учење. Ова платформа омогућава организацијама сигурно, скалабилно и високо перформантно окружење за покретање апликација, управљање радним оптерећењима и обраду великих количина података.
- *Insomnia – REST API* клијент, за организовање, складиштење и извршавање *API* захтева.
- *RabbitMQ* – софтвер за асинхрону комуникацију између сервиса. Има улогу средњег слоја приликом слања порука са једног сервиса на други.
- *gRPC* – технологија високе поузданости и перформанси за синхрону комуникацију између сервиса. Користи *RPC* протокол за размену порука.
- *JSON Web Token* – отворени стандард који дефинише правила преноса информација између страна у облику *JSON* објекта.
- *Docker* – отворена платформа за развој, слање и покретање апликација. *Docker* омогућава одвајање софтвера од инфраструктуре у којој је направљен, што омогућава слање и покретање у другим инфраструктурама и олакшава пуштање софтвера у продукцију.
- *Kubernetes* – систем отвореног типа за аутоматизацију процеса менаџмента, скалирања и пуштања у продукцију контејнеризованих апликација. Групише контејнере који заједно чине логичку јединицу, систем или апликацију, што је од посебног значаја у контексту микро-сервиса.
- *Horizontal Pod Autoscaler (HPA)* – компонента *Kubernetes* платформе која омогућава динамичко прилагођавање броја инстанци (*pod*-ова) на основу метрика оптерећења.
- *Prometheus* – отворени систем за прикупљање, чување и обраду метрика из различитих

компоненти дистрибуираних система. Развијен је првенствено за потребе мониторинга cloud и *container-based* апликација, а данас представља стандардно решење за надзор *Kubernetes* окружења.

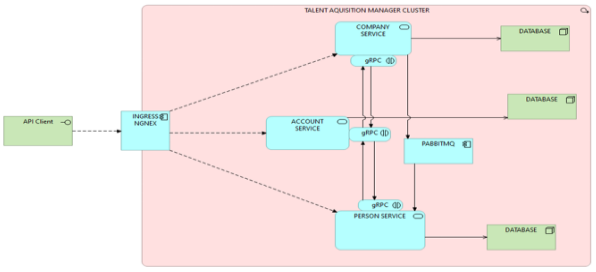
- *Grafana* – отворена платформа намењена за визуализацију, анализу и праћење метрика и података из различитих извора. Најчешће се користи у комбинацији са *Prometheus*-ом. Главна улога *Grafana*-е је да прикупљене метрике и логове прикаже у виду интерактивних графова, табела и *dashboard*-а, чиме се олакшава праћење перформанси система и правовремено уочавање проблема.
- *Google Cloud SQL* – managed релациона база података у *cloud*-у.
- *K6* – отворени алат за *load* и *performance testing*.

4.3. Архитектура апликације

За потребе рада, креиране су две верзије апликације, логички потпуно исте и пружају потпуно исте могућности. Једна верзија је микро-сервисна, која је помогла да се на практичном примеру испита *cloud-native* приступ *cloud* окружењу, а друга монолитна, која је помогла да се на практичном примеру истражи „*replatform*“ приступ миграцији постојеће апликације.

4.4. Микро-сервисна архитектура апликације

Архитектура се састоји од три микро-сервиса: *AccountService*, *CompanyService* и *PersonService*. Поред ових компоненти, саставни део апликације чини и *Ingress NGINX* контролер, који је задужен за балансирање оптерећења и даље рутирање *API* захтева према сервисима.
Сваки од сервиса има своју *Google Cloud SQL* базу података која трајно складишти податке потребне за рад сервиса. Поред горе наведених компоненти, саставни део апликације чини и *RabbitMQ* сервис за асинхрони пренос порука. Сваки од сервиса користи *gRPC* технологију за синхрону комуникацију са другим сервисима (*слика 1*).



Слика 1. Приказ архитектуре микро-сервисне апликације

AccountService је микро-сервис који обавља сву пословну логику везану за корисничке налоге корисника ове апликације.
CompanyService и *PersonService* представљају два одвојена микро-сервиса унутар нашег система. Ови сервиси су специјализовани за ефикасно управљање профилима корисника на нашој апликацији.

4.5. Монолитна архитектура апликације

Ова верзија апликације је настала спајањем сва три сервиса микро-сервисне верзије у један заједнички монолитни сервис, који омогућава све логичке функционалности микро-сервисне апликације. Оваквим приступом из архитектуре се избацује доста компоненти микро-сервисне верзије.
Три базе података су замењене једном, која треба да чува све податке. *Ingress* контролер систему више није потребан јер сав саобраћај сада улази у једну тачку, тј. заједнички сервис. За комуникацију између различитих логичких целина користе се апликацијски интерфејси, што уклања потребу за *gRPC* синхроном и *RabbitMQ* асинхроном комуникацијом.
Монолитна архитектура одбацује *cloud-native* приступ и представља добар пример *legacy* апликације, што омогућава симулацију миграције на *cloud*, а посебно „*Replatform“* приступ.

5. ПОСТАВЉАЊЕ АПЛИКАЦИЈЕ У *CLOUD* ОКРУЖЕЊЕ

За потребе овог рада коришћен је *gCloud*, платформа компаније *Google*. Обе архитектуре (микро-сервисна и монолитна) постављене су у засебне кластере, како би испитивања оптерећења и потрошње ресурса на њима била релевантнија. Обе апликације користе *e2-medium node pool*, где свака инстанца унутар *pool*-а пружа 2 виртуелна процесорска језгра и 4GB RAM меморије.
Код микро-сервисне архитектуре, pod сваког сервиса је иницијално покренут у једној инстанци, а максимална вредност хоризонталног скалирања код сваког сервиса је постављена на три. Подови се скалирају када оптерећење ресурса којима располаже *pod* (CPU језгара и RAM меморије) достигне 80%. Унутар кластера је постављен *RabbitMQ message broker*, већ поменути *HPA* (*horizontal pod autoscaling*), као и алати за праћење и визуализацију метрика, *Prometheus* и *Grafana*. За складиштење података користи се *Google Cloud SQL*, који за потребе овог система садржи три базе података.
Код монолитне архитектуре, апликација је иницијално покренута на једној инстанци, а максимална вредност хоризонталног скалирања је постављена на три. Као и код под-ова у оквиру кластера микро-сервисне апликације, под-ови се скалирају ако оптерећење ресурса којима располаже pod достигне 80%. За разлику од кластера микро-сервисне архитектуре, кластер монолитне апликације не садржи *RabbitMQ* инстанцу, и систем користи само једну базу података.
Кластер микро-сервисне апликације користи видно више ресурса када је систем у стању без оптерећења или стању лаког оптерећења. Тада микро-сервисни кластер користи четири инстанце *e2-medium node*-а, док монолитни кластер користи три. Треба узети у обзир да на оба кластера по један *node* нема скоро никакво оптерећење, тако да је реална искоришћеност два *node*-а за монолитну, а три *node*-а за микро-сервисну архитектуру. Овој, на изглед превеликој оптерећености ресурса, доста доприносе и инстанце које раде мониторинг постојећег система и праве

метрике, логове или воде рачуна о хоризонталном скалирању система.

6. ОПТЕРЕЋЕЊЕ СИСТЕМА И РЕЗУЛТАТИ ТЕСТИРАЊА

За потребе тестирања коришћен је K6 софтверски алат, за load и performance тестирање web апликација. За потребе овог рада изведени су load (реално оптерећење) и stress (високо оптерећење) тестови. У овим тестовима, сваки виртуелни корисник прати кораке редоследом: креирање налога, креирање профила на налогу, измена профила, брисање профила, брисање налога.

6.1. *Load* тестови система

Оба система су тестирана за случај нормалног оптерећења. За ово тестирање коришћено је 40 виртуелних корисника који су у исто време, у трајању од 15 минута, користили апликацију, извршавајући горе наведене кораке. Добијени су резултати видљиви на слици 2.

Metric	Monolithic	Microservices
Total requests	147,833	108,801
Average request duration	93.47 ms	188.28 ms
Median (p50)	63.48 ms	69.1 ms
p90 latency	175.42 ms	301.57 ms
p95 latency	222.54 ms	526.92 ms
Max response time	30.1 s	21.9 s
Error rate	0.13%	0.00%
Average iteration duration	1.7 s	2.31 s

Слика 2. Резултати *load* тестирања система

6.2. *Stress* тестови система

Оба система су такође тестирана и за случај неочекиваног преоптерећења. За ово тестирање коришћено је до 200 виртуелних корисника који су у исто време, 16 минута, користили апликацију, извршавајући горе наведене кораке. Број корисника је временом постепено повећаван са 40 на 200.

Metric	Monolithic	Microservices
Total requests	207,094	102,135
Average request du.	398.7 ms	991.88 ms
Median (p50)	178.89 ms	77 ms
p90 latency	860.26 ms	495.32 ms
p95 latency	1.04 s	2.97 s
Max response time	2 min 40s	46.68 s
Error rate	0.07%	0.008%
Average iteration dura.	3.89 s	7.94 s

Слика 3. Резултати *stress* тестирања система

Добијени су резултати видљиви на слици 3.

7. ЗАКЉУЧАК

На конкретном примеру мале апликације, на основу резултата тестова, упоређене су перформансе при реалном и високом оптерећењу. Резултати показују да „*replatform*“ миграција, симулирана кроз монолитну архитектуру са једним сервисом, једном базом података и *load balancer*-ом, пружа бољу ефикасност у контексту мале до средње апликације. Монолитна верзија обрадила је значајно више захтева са нижом просечном латенцијом и краћим временом итерације виртуелних корисника, што указује на мањи *overhead* код монолитне верзије.

С друге стране, *cloud-native* приступ, имплементиран кроз микро-сервисну архитектуру, са више сервиса, *gRPC* комуникацијом и *RabbitMQ*-ом, показао је бољу отпорност на грешке и бољу медијану латенције. Међутим, већа просечна латенција и варијабилност потврђују познати *overhead* микро-сервиса, посебно у малим системима где мрежна комуникација и координација превазилазе потребе.

Cloud-native приступ, иако супериоран у скалабилности и агилности за велике, дистрибуиране системе, показује мање предности у овом сценарију, посебно уз ограничене ресурсе и не тако логички и функционално сложеним системом.

8. ЛИТЕРАТУРА

[1] Fahmideh, Low, Beydoun & Daneshgar , “Cloud Migration Process: A Survey Evaluation Framework and Open Challenges“, 2016.

[2] https://aws.amazon.com/what-is/cloud-native/ (pristupljeno u oktobru 2025.)

[3] www.netapp.com/blog/aws-cvo-blg-strategies-for-aws-migration-the-new-7th-r-explained/ (pristupljeno u oktobru 2025.)

[4] www.techtarget.com/searchcloudcomputing/tip/The-Rs-of-cloud-migration-How-to-choose-the-right-method (pristupljeno u oktobru 2025.)

Кратка биографија:

Цвијетин Глишић рођен је 2000. године у Бијељини. Дипломирао је 2023. године, на смеру Примењено софтверско инжењерство.

Контакт: cvijetinglisic@gmail.com