Зборник радова Факултета техничких наука, Нови Сад

УДК: 004:007
ДОИ: https://doi.org/10.24867/35OI04Andric

Упоредна анализа микросервисне и serverless архитектуре у дизајну дистрибуираних система

Comparative Analysis of Microservices and Serverless Architectures in Distributed System Design

Валентина Андрић, *Факултет техничких наука, Нови Сад*

Област – ИНЖЕЊЕРСТВО ИНФОРМАЦИОНИХ СИСТЕМА

Кратак садржај – *Овај рад анализира савремене архитектуре за развој дистрибуираних система, са посебним фокусом на микросервисну и serverless архитектуру. Приказане су њихове основне карактеристике, начин функционисања, као и предности и недостаци у различитим сценаријима примене. Кроз упоредну анализу разматрају се кључни критеријуми избора, као што су скалабилност, трошкови, одржавање и сложеност система. Циљ рада је да пружи јасне смернице за избор одговарајуће архитектуре у зависности од пословних и техничких захтева савремених софтверских система.*

Кључне речи: *Микросервисна архитектура, serverless архитектура, дистрибуирани ситеми, скалабилност, перформансе*

Abstract – *This paper analyzes modern architectures for the development of distributed systems, with a particular focus on microservice and serverless architectures. Their fundamental characteristics, operating principles, as well as advantages and disadvantages in various application scenarios are presented. Through a comparative analysis, key criteria for architecture selection are examined, including scalability, cost, maintenance, and system complexity. The goal of this paper is to provide clear guidelines for selecting an appropriate architecture based on the business and technical requirements of modern software systems.*

Keywords: *Microservice architecture, serverless architecture, distributed systems, scalability, performance*

НАПОМЕНА: Овај рад је проистекао из матер рада чији ментор је био доцент др Марко Арсеновић.

1. УВОД

Савремени софтверски системи постају све сложенији, са растућим обимом података и захтевима за високом доступношћу, перформансама и скалабилношћу. Како апликације еволуирају, њихова архитектура се мења од једноставних монолита до комплексних микросервисних и serverless система. Избор одговарајуће архитектуре кључно утиче на перформансе, одржавање и развој система. Док су монолитни системи имали ограничења у скалабилности и флексибилности, сервисно оријентисане архитектуре донеле су већу модуларност. Данас су доминантни приступи микросервисна и serverless архитектура, који се разликују по начину дефинисања граница система, расподели одговорности и дизајну инфраструктуре. Рад се фокусира на упоредну анализу ова два приступа, истичући предности, недостке и применљивост у различитим ситуацијама, уз теоријски преглед и студију случаја.

2. ТЕОРИЈСКЕ ОСНОВЕ

Ово поглавље даје сажет преглед еволуције апликационих архитектура и основних концепата дистрибуираних система, као теоријску основу за разумевање савремених архитектонских приступа. Развој софтверских система условљен је растућом сложеношћу пословних захтева, потребом за већим перформансама и ефикаснијим скалирањем, што је довело до постепеног превазилажења једноставних архитектонских модела и увођења флексибилнијих решења. Слика 1 илуструје еволутивни развој софтверских архитектура о чему ће бити речи у наставку.

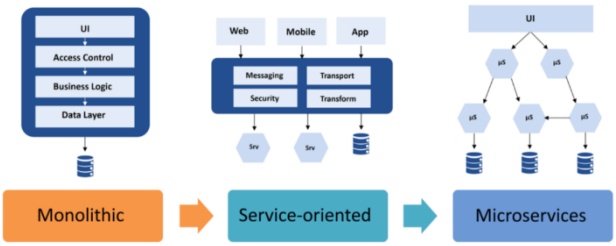
Монолитна архитектура представља полазну тачку у развоју већине апликација, јер омогућава једноставну имплементацију и управљање у почетним фазама. Међутим, како системи расту, јављају се ограничења у погледу одржавања, скалабилности и тимског рада [2]. Као еволутивни корак, развијени су модуларни монолит и вишеслојна архитектура, који уводе јаснију организацију кода и сепарацију одговорности, чиме се постављају темељи за дистрибуиране системе.

Даљи развој довео је до појаве сервисно оријентисане архитектуре *event-driven* и микросервисне архитектуре, које омогућавају поделу система на мање, логички независне компоненте које комуницирају преко мреже. Ови приступи доносе већу скалабилност, флексибилност и отпорност на грешке, али истовремено уводе већу сложеност у погледу комуникације, мониторинга и управљања системом. Serverless архитектура са друге стране представља највиши ниво апстракције, при чему је инфраструктура

у потпуности препуштена *cloud* провајдерима, што додатно поједностављује развој, али намеће нова ограничења.

Микросервисна архитектура представља еволуцију SOA (*Service-Oriented Architecture)* концепта, у којој је систем подељен на мале, независне сервисе са јасно дефинисаним одговорностима и сопственим базама података. Овакав приступ омогућава лакши развој, тестирање и дистрибуцију сервиса, бољу скалабилност и већу отпорност на грешке, али уводи већу сложеност у управљању и одржавању система.

Serverless архитектура представља виши ниво апстракције, где cloud провајдер у потпуности управља инфраструктуром, а апликациона логика се извршава кроз функције на захтев FaaS (*Function-as-a-Service*). Овај модел омогућава аутоматско скалирање и ефикасније коришћење ресурса, али уз одређена ограничења у контроли окружења и управљању стањем система.



Слика 1. Приказ еволуције софтверских архитектура [3]

2.1. Основни концепти дистрибуираних система

Дистрибуирани системи се заснивају на сарадњи више међусобно повезаних компоненти које за корисника функционишу као јединствена целина. Њихове кључне карактеристике су скалабилност, поузданост и транспарентност, као и потреба за компромисима у складу са CAP (*Consistency, Availability, Partition Tolerance)* теоремом [4]. Иако уносе додатну сложеност, дистрибуирани системи омогућавају бољу организацију тимова, већу отпорност на отказе и лакше прилагођавање променљивим захтевима, због чега представљају основу савремених софтверских архитектура о чему ће бити речи даље у овом раду.

3. МІКРОСЕРВИСИ

Микросервисна архитектура последњих година постаје један од доминантних приступа у развоју савремених софтверских система, како за нове апликације, тако и за модернизацију постојећих решења. Основна идеја овог приступа јесте декомпозиција сложених система на мање, логички независне сервисе који реализују јасно дефинисане пословне функционалности и међусобно комуницирају преко мреже. Једна од кључних предности микросервисне архитектуре јесте јасна подела одговорности и висок степен аутономије појединачних сервиса. Сваки микросервис се може независно развијати, тестирати, распоређивати и скалирати, што омогућава брже увођење промена и бољу отпорност система на отказе. Независно управљање подацима по сервису повећава флексибилност и смањује међузависност компоненти, док асинхрони комуникациони обрасци доприносе слабој спрегнутости и бољој скалабилности система [8]. Микросервиси такође омогућавају технолошку хетерогеност, односно избор најпогоднијих програмских језика, база података и алата у складу са специфичним захтевима појединих сервиса. Овај приступ позитивно утиче на перформансе и одрживост система, као и на организацију тимова, који се могу структурирати око појединачних сервиса и радити паралелно, са јасно дефинисаним одговорностима.
Са друге стране, микросервисна архитектура уводи значајну сложеност у дизајну и одржавању система. Управљање дистрибуираним компонентама, комуникацијом и отказима захтева пажљиво планирање и добро познавање домена. Одржавање конзистентности података представља посебан изазов, јер се системи често ослањају на модел смањене конзистенције, што захтева додатне механизме синхронизације и обраде изузетака.

Поред тога, микросервисни системи обично носе веће инфраструктурне и организационе трошкове, као и повећане захтеве у погледу безбедности и *DevOps* пракси. Због тога микросервиси нису увек оправдан избор, нарочито за мање системе и тимове, где једноставнији архитектонски приступи могу бити ефикаснији. Успешна примена микросервисне архитектуре захтева искусне тимове, адекватне алате и јасно дефинисане архитектонске принципе.

4. *SERVERLESS* АРХИТЕКТУРА

Serverless архитектура представља савремени приступ развоју апликација у којем је управљање инфраструктуром у потпуности препуштено cloud провајдеру, а ресурси се користе по потреби. Главна предност овог модела је поједностављење процеса развоја, тестирања и извршавања апликација, јер програмери не морају да се баве конфигурацијом и одржавањем сервера. Поред тога, модел наплате по потрошњи омогућава оптимизацију трошкова, док аутоматско скалирање прилагођава ресурсе тренутном оптерећењу, обезбеђујући висок ниво флексибилности и доступности система. Serverless архитектура такође омогућава фокус на развој пословне логике, изолацију појединачних функција и лакше управљање ресурсима, што је посебно погодан приступ за стартап компаније, виртуелне асистенте и *chatbot*-е са непредвидивим оптерећењем [5].

Са друге стране, ова архитектура доноси и одређене изазове. Примена serverless решења ствара зависност од конкретног cloud провајдера, што отежава миграцију на другу платформу. Усвајање овог приступа може бити комплексно за тимове без претходног искуства, нарочито због специфичних концепата као што су дебаговање и управљање отказима. Serverless је најпогоднији за краткотрајне и брзе задатке, док дуготрајни и комплексни процеси могу бити ограничени. Такође, у IoT (*Internet of Things*) апликацијама поузданост система у великој мери зависи од стабилности интернет конекције, што може

представљати додатни изазов. Serverless архитектура доноси значајне предности у смислу флексибилности и трошкова, али њена примена мора бити добро планирана и прилагођена специфичним захтевима апликације.

5. МИКРОСЕРВИСИ НАСПРАМ SERVERLESS АРХИТЕКТУРЕ

Табела 1. Поређење serverless и микросервисне архитектуре

Карактеристика	**Serverless**	**Микросервиси**
Архитектура	Апликација од функција на захтев	Апликација од независних сервиса
Управљање инфраструктуром	Cloud провајдер	Тим (сервери, контејнери, оркестрација)
Deployment јединица	Функција	Сервис
Скалирање	Аутоматско	Ручно или помоћу алата
Трошкови	Плаћање по извршавању	Плаћање за стално активну инфраструктуру
Стање (state)	Углавном stateless	Stateless или stateful
Перформансе	Може бити кашњење при првом позиву (cold start)	Стабилне
Контрола окружења	Ограничена	Потпуна
Комплексност	Мања	Већа, захтева DevOps знање
Зависност од трећих страна	Више ослањања (BaaS)	Слободан избор
Погодно за апликације	Мале и средње	Велике, комплексне

Табела 1 описује поређење serverless и микросервисне архитектуре са аспекта скалабилности, трошкова, сложености имплементације и начина управљања ресурсима.

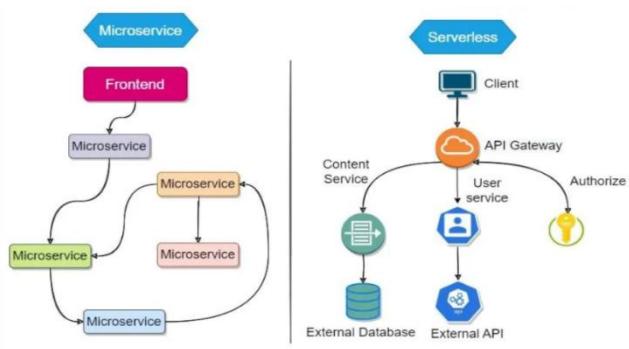
Serverless архитектура као што је већ речено заснива се на извршавању апликационе логике као функција које се активирају догађајима, при чему cloud провајдер управља инфраструктуром, скалирањем, безбедношћу и мониторингом. Клијентска апликација комуницира преко API (*Application programming interface*) *Gateway*-а, који рутира захтеве ка одговарајућим функцијама (нпр. сервис за кориснике или садржај) као што се може видети на слици 2.

Микросервисна архитектура, са друге стране, користи дуготрајно активне сервисе који могу да задрже стање, кеширају податке и имају трајне конекције ка базама података (слика 2). Континуирано праћење перформанси и доступности омогућава стабилније извршавање сложенијих захтева.

Serverless омогућава аутоматско и тренутно скалирање по потреби, погодан је за апликације са променљивим оптерећењем (нпр. онлајн резервације карата), а плаћање је по коришћеном ресурсу (*„pay-as-you-go“*). Микросервисна архитектура омогућава пуну контролу инфраструктуре, стабилне перформансе без *cold start*-а и захтева DevOps знање за управљање и скалирање сервиса [7].

Serverless функције су углавном *stateless*, док микросервиси могу бити и stateless и *stateful,* што омогућава праћење сесија, кеширање и дуготрајне процесе. Serverless је мање комплексна на почетку развоја и погодна је за мање и средње апликације, док микросервиси захтевају веће искуство, али су погодни за велике и дугорочне системе.

Serverless се ослања на cloud провајдера и BaaS (*Business-as-a-service*) услуге, што може ограничити избор алата, док микросервисна архитектура омогућава слободан избор зависности и већи степен контроле.



Слика 2. Упоредни приказ микросервисне и serverless архитектуре [1]

5.1. Хибридни приступ

Хибридни приступ користи микросервисе за дуготрајне и критичне делове система, а serverless функције за једноставне, догађајима покренуте задатке (нпр. обрада фајлова, слање нотификација, периодични задаци). Алати као *Logic Apps* или *Step Functions* омогућавају тригере и интеграцију микросервиса као функција. *Netflix, Amazon, Google, Spotify* и *Uber* су примери где се

користи ова комбинација за максималну флексибилност и оптимизацију ресурса.

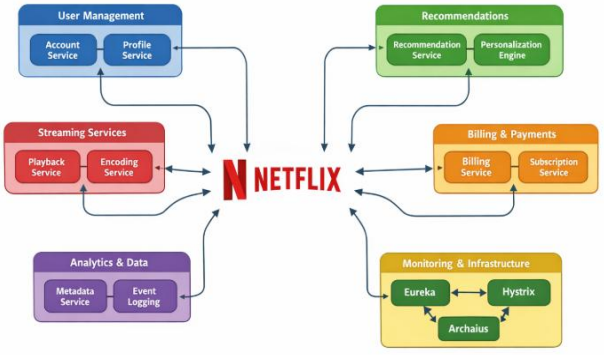
6. СТУДИЈА СЛУЧАЈА: NETFLIX

Netflix је у раним годинама радио као монолитна апликација, где је цела пословна логика била у једном систему. Такав приступ је функционисао док је систем био мали, али са растом броја корисника и захтева за новим функцијама, ограничења су постала видљива. Преласком на микросервисну архитектуру (слика 3), појединачни сервиси могу отказивати изоловано, без утицаја на целокупно корисничко искуство, а независни развој и deploy сервиса убрзали су циклус развоја [6]. Netflix ради на AWS платформи, користећи сервисе као што су:

- **EC2** (*Elastic Compute Cloud)* – за захтевне обраде и енкодирање видео садржаја,
- **S3** (*Simple Storage Service*)– за складиштење садржаја и метаподатака,
- ***DynamoDB и Cassandra*** – NoSQL базе за корисничке податке,
- **AWS Lambda** (*Amazon Web Services Lambda*) – за serverless функције (обрада логова, анализа података),
- **ELB** (*Elastic Load Balancing*) – за расподелу саобраћаја,
- ***Kinesis и Kafka*** – за стримовање и обраду података у реалном времену.

Клијентски слој обухвата мобилне апликације, паметне телевизоре и веб прегледаче, који шаљу захтеве преко API Gateway-а ка микросервисима. Слој микросервиса се извршава на *Titus* платформи, што омогућава независно скалирање и изолацију грешака.

Слој података користи Cassandra, S3 и Kafku за обраду великих количина података са високом доступношћу и ниском латенцијом. Слој за праћење укључује *Atlas* (метрике), *Spinnaker* CI/CD (*Continuous Integration/Continuous Delivery)* и примену *Chaos Engineering* приступа, укључујући алат *Chaos Monkey* који намерно изазива отказе како би се тестирала отпорност система и обезбедила континуирана стабилност [6].



Слика 3. Приказ архитектуре Netflix-а на високом нивоу

7. ЗАКЉУЧАК

У овом раду анализиране су микросервисна и serverless архитектура као савремени приступи развоју дистрибуираних система. Кроз упоредну анализу показано је да избор архитектуре зависи од карактеристика система, оптерећења и потребе за контролом над инфраструктуром, те да не постоји универзално решење.

Истакнут је значај правилног архитектонског дизајна у раним фазама развоја, јер он директно утиче на скалабилност, одрживост и трошкове система. Такође, закључује се да ове архитектуре могу бити комплементарне, при чему њихова комбинација омогућава већу флексибилност и ефикасније коришћење ресурса.

Будућа истраживања могу бити усмерена на практичну имплементацију и детаљније поређење перформанси у реалним условима.

8. ЛИТЕРАТУРА

[1] Arslan Ahmad: "*Monoliths vs Microservices vs Serverless*: Choosing the Right Architecture", 2023.

[2] Sandra Đuričin: *Migracija monolitnog sistema na mikroservisnu arhitekturu*, 2022.

[3] Shayesteh Arfa Mahdi, "Microservices vs SOA", 2023.

[4] George Coulouris, Jean Dollimore, Tim Kindberg, Gordon Blair, *Distributed Systems: Concepts and Design*, Addison-Wesley, 2012.

[5] Lizheng Jiang, Yunman Pei, Jiantao Zhao, "Overview of Serverless Architecture Research", 2019.

[6] Kovvuru Ismail: *How Netflix Designed Its Global Cloud Architecture on AWS: The Real Reason Netflix Moved to AWS*. November 11, 2025.

[7] Eric Brewer: *Towards Robust Distributed Systems*, Proceedings of the ACM Symposium on Principles of Distributed Computing, 2000.

[8] Sam Newman: *Building Microservices*, O'Reilly Media, 2015.

Кратка биографија:



Валентина Андрић рођена је у Прибоју 2000. год. Дипломирала на Факултету техничких наука, на смеру Инжењерство информационих система, 2023 год.