Izzivi in vzorci zasnove mikrostoritev v brezstrežniškem okolju

Tilen Hliš, Luka Pavlič

Univerza v Mariboru, Fakulteta za elektrotehniko, računalništvo in informatiko, Maribor, Slovenija
tilen.hlis@um.si, luka.pavlic@um.si

Sinopsis Mikrostoritvena arhitektura je preizkušen, in kot kažejo podatki, tudi vodilen arhitekturni stil informacijskih rešitev zadnjega desetletja. Kot je praksa na vseh inženirskih področjih, tudi pri vpeljavi mikrostoritvene arhitekture uporabljamo preizkušene dobre prakse. Načrtovalski vzorci ne predstavljajo konkretne implementacije, temveč praktično preizkušene ideje, kako rešiti določeno skupino izzivov. Vzorce lahko umestimo v kategorije ali skupine - vsaka izmed njih naslavlja določeno problematiko s katero se soočajo razvojne skupine. Brezstrežniško računalništvo z vedno hitrejšim razvojem oblačnega računalništva vedno bolj pridobiva na pomenu. Arhitekture brez strežnika predvidevajo zasnovo informacijskih rešitev, ki vključuje zaledje kot storitev (angl. Backend as a Service - BaaS) in kodo po meri (angl. Functions as a Service -FaaS), ki se izvaja v upravljanih, kratkotrajnih vsebnikih na izbrani platformi.

Ključne besede:
mikrostoritvena arhitektura
brezstrežniško računalništvo
načrtovalski vzorci

Univerzitetna založba Univerze v Mariboru
ISBN 978-961-286-639-6
DOI https://doi.org/10.18690/um.feri.10.2022.16

1 Uvod

Področje storitveno usmerjene arhitekture in računalništva v oblaku je v zadnjih 10 letih doživelo evolucijo pri razvoju, uvajanju in vzdrževanju informacijskih sistemov. Pojem storitveno usmerjene arhitekture, ki zajema spletne storitve opredeljuje, da so poslovna pravila aplikacij vgrajena v storitve same, komunicirajo pa preko standardnega protokola. Omenjen način omogoča enotnost pri zagotavljanju, iskanju, interakciji in uporabi storitev. Domenska področja podjetja s poslovnimi procesi lahko prenesemo na posamezno storitev. Osrednji namen uporabe storitveno usmerjene arhitekture je integracija nizko sklopljenih neodvisnih aplikacij, pri čemer posamezne gradnike aplikacij strukturiramo kot storitve. Na trgu se pojavlja vedno več spletnih storitev, ki ne podpirajo vseh načel storitveno usmerjene arhitekture (tj. pogodba o storitvi, nizka sklopljenost, ponovna uporaba, abstrakcija, delovanje brez stanja, široka združljivost, odkrivanje in interoperabilnost storitev) in s tem se je pojavil nov pojem, ki je v svetu IT znan kot mikrostoritvena arhitektura. [1, 2]

Mikrostoritvena arhitektura je pomembnejši načrtovalski slog v zadnjem desetletju. Mikrostoritve izhajajo iz problematike spletnih storitev, ki slabo ali sploh ne podpirajo vseh načel storitveno usmerjene arhitekture. Številne prednosti mikrostoritvene arhitekture vključujejo nizko sklopljenost, neodvisnost posameznih storitev in omogočeno razširljivost, ponujajo sisteme z višjimi zmogljivostmi, boljšim delovanjem, organizacijsko pa ponujajo možnost hkratnega neodvisnega razvoja posameznih gradnikov informacijskih rešitev. Arhitekturni stil se je razvijal vzporedno z razvojem programsko usmerjene industrije in tako je uporaba mikrostoritvene arhitekture v praksi v preteklosti prehitela akademsko sfero. Nastala vrzel se, zahvaljujoč številnim raziskavam, povezanimi z mikrostoritvami s področja novih metodologij, procesov in orodij, zmanjšuje [3]. Zaradi posebnosti in odstopanj od uveljavljenih dobrih praks je potrebno za vpeljavo mikrostoritvene arhitekture uporabiti številne inovativne pristope pri naslavljanju nefunkcionalnih zadev. Kljub vsemu lahko dobre prakse že najdemo v obliki zbirk in katalogov. Načrtovalski vzorci ne predstavljajo konkretne implementacije, temveč praktično preizkušene ideje, kako rešiti določeno skupino izzivov. Vzorce lahko umestimo v kategorije ali skupine - vsaka izmed njih naslavlja določeno problematiko s katero se soočajo razvijalci . Eno izmed področij, ki je povezano s svojo skupino načrtovalskih vzorcev mikrostoritvene arhitekture, je področje izvajanja mikrostoritev. Skupina zajema vzorce, ki so povezani z izvajanjem posameznih mikrostoritev in celotne informacijske rešitve, pri čemer izpostavljamo vzorec, ki temelji na uvedbi brez strežnika (angl. Serverless deployment) [4]. Vzorec je oblikovan okoli skrivanja osrednjih strežniških konceptov. Uporabljena infrastruktura sprejme kodo in jo požene. Vzorec je zanimiv, saj predstavlja povezavo med sicer samosvojima stiloma arhitekturne zasnove, mikrostoritveno in brezstrežniško.

Brezstrežniško računalništvo z hitrim razvojem oblačnega računalništva vedno bolj pridobiva na pomenu. Arhitekture brez strežnika predvidevajo zasnovo informacijskih rešitev, ki vključuje zaledje kot storitev (angl. *Backend as a Service - BaaS*) in kodo po meri (angl. *Functions as a Service -FaaS*), ki se izvaja v upravljanih, kratkotrajnih vsebnikih na izbrani platformi [5].

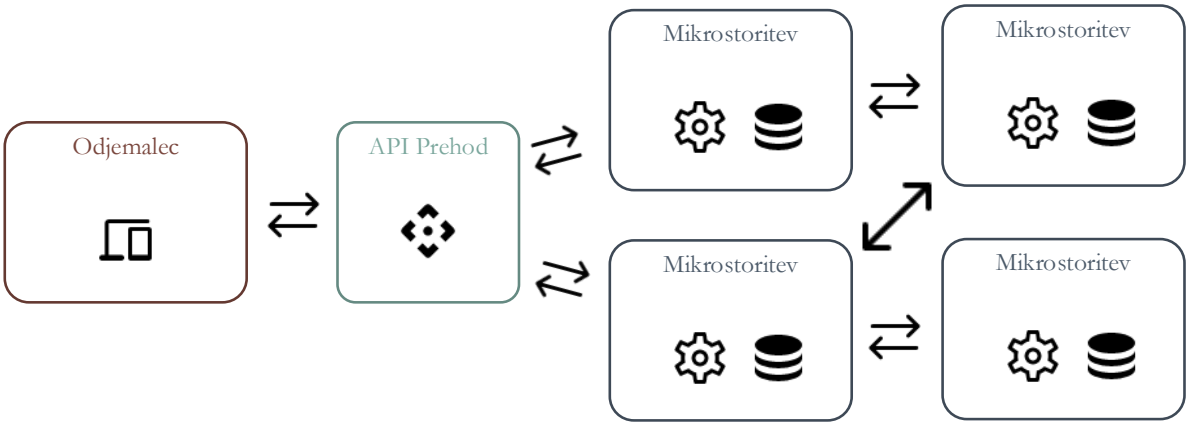
2 Mikrostoritvena arhitektura in brezstrežniško računalništvo

Brezstrežniško računalništvo ima več skupnih značilnosti z mikrostoritvenim stilom, a tudi nekaj pomembnih razlik. V nekaterih primerih je za delo bolj primerna uporaba sistema brez strežnika kot raba mikrostoritev. Da bi razumeli, katero izmed arhitekturnih zasnov uporabiti, morajo razvijalci upoštevati, kako se ta dva pristopa dopolnjujeta in predvsem razlikujeta.

2.1. Mikrostoritvena arhitektura

Trendi nakazujejo, da je uporaba mikrostoritvene arhitekture pogosta izbira pri snovanju sodobnih informacijskih rešitev [3]. Arhitekturni stil izhaja iz storitveno usmerjene arhitekture in tako omogoča skladno integracijo med seboj neodvisnih sistemov. Z nekaterimi prej znanimi in uporabljenimi arhitekturnimi zasnovami smo

informacijsko rešitev gradili v smislu t. i. monolita, ob upoštevanju smernic mikrostoritvene arhitekture pa gradimo množico neodvisnih mikrostoritev. Mikrostoritvena arhitektura sledi načelu, da morajo biti storitve majhne in lahke, pri čemer naj vsaka storitev opravlja samo eno poslovno nalogo. Vsaka storitev naj ima svojo bazo podatkov, kar še dodatno zmanjša povezanost med njimi. Celoten sistem je sestavljen iz več manjših, šibko sklopljenih delov, kar nam omogoča neodvisno spreminjanje vsakega dela posebej. Omogočena razširljivost ponuja sisteme z višjimi zmogljivostmi in boljšim delovanjem. Neodvisnost in nizka sklopljenost gresta pri mikrostoritveni arhitekturi preko meja izvorne kode in se nanašata na izvajalno okolje ter razvojno ekipo, ki skrbi za razvoj določene mikrostoritve. Slika 1 prikazuje shematski prikaz tipične zasnove mikrostoritvene rešitve.



Slika 1: Mikrostoritvena arhitektura.
Vir: lasten.

Čeprav so načela in smernice tega arhitekturnega stila znane že dalj časa, sta eno izmed najbolj pogosto uporabljenih definicij mikrostoritvene arhitekture šele leta 2014 zapisala Martin Fowler in James Lewis [2]. Definicijo lahko povzamemo kot: »Arhitekturni slog mikrostoritev je pristop k razvoju ene same aplikacije kot zbirke majhnih storitev, od katerih vsaka deluje v svojem procesu in komunicira s preostalimi storitvami s pomočjo določenih mehanizmov«. Storitve so zgrajene na podlagi poslovnih zmogljivosti in jih je mogoče neodvisno uvesti s popolnoma avtomatiziranimi stroji za uvajanje. Obstaja minimalno centralizirano upravljanje teh storitev, ki so lahko napisane v različnih programskih jezikih in uporabljajo različne tehnologije za shranjevanje podatkov. [1, 2, 3]

Posamezne mikrostoritve tipično ne vzdržujejo stanja. Zahteve in odgovori se med storitvami tipično prenašajo preko protokola HTTP, pri čemer ponudijo odgovore v formatu JSON (angl. JavaScript Object Notation) ali XML (angl. Extensible Markup Language). Končne točke izpostavljene s strani mikrostoritev ponujajo rabo metod protokola HTTP (za komunikacijo uporabljajo princip REST (angl. Representational state transfer)) in predstavljajo povezavo s poslovno logiko sistema. Asinhrono komunikacijo lahko dosežemo preko protokolov, kot je MQTT (angl. MQ Telemetry Transport); tako lahko pošiljamo in pridobivamo podatke iz različnih senzorjev. Uveljavljen sinhroni stil komunikacije RPC (oddaljen klic procedur, angl. Remote Procedure Call) je poleg uporabe HTTP vedno pogosteje implementiran tudi z ogrodji, kot je gRPC. Le-ti že privzeto ponujajo možnost asinhronega izvajanja klicev in za delovanje uporabljajo HTTP/2. Posamezne storitve v rešitvi, ki uporablja mikrostoritveni arhitekurni stil lahko med seboj na asinhorni način komunicirajo tudi z uporabo sporočilnih sistemov, kot so RabbitMQ, ActiveMQ in Apache Kafka. Zaradi arhitekturne zasnove mikrostoritve dopolnjujejo agilen razvoj in uporabo inženirskih praks DevOps. [1, 3]

Ena izmed prednosti mikrostoritvenih sistemov v primerjavi z uporabo monolitne zasnove sistema je odpornost. Kadar odpove eden izmed gradnikov sistema, to ne pomeni izpada celotnega sistema. Možnost lažje skalabilnosti rešuje morebitne izzive ob višji obremenitvi ene, več ali vseh storitev sistema. V izogib upočasnjenega delovanja in neodzivnosti lahko s pomočjo uporabe določenih tehnologij prilagodimo skalabilnost le storitvam, ki to dejansko potrebujejo. Po drugi strani pa predstavlja skalabilnost v sistemih z monolitno zasnovo precejšen izziv. Omeniti

velja, da je lahko kratkoročno gledano vpeljava mikrostoritev dražja, pri čemer je v zakup potrebno vzeti druge prednosti, ki nam olajšajo delo ter dolgoročno prihranijo čas in denar. [6, 7]

Na podlagi prej podane definicije in do sedaj zapisanega lahko izpeljemo sledeče lastnosti, ki bi jih naj imeli informacijski sistemi, ki uporabljajo mikrostoritveno arhitekturo [4]:

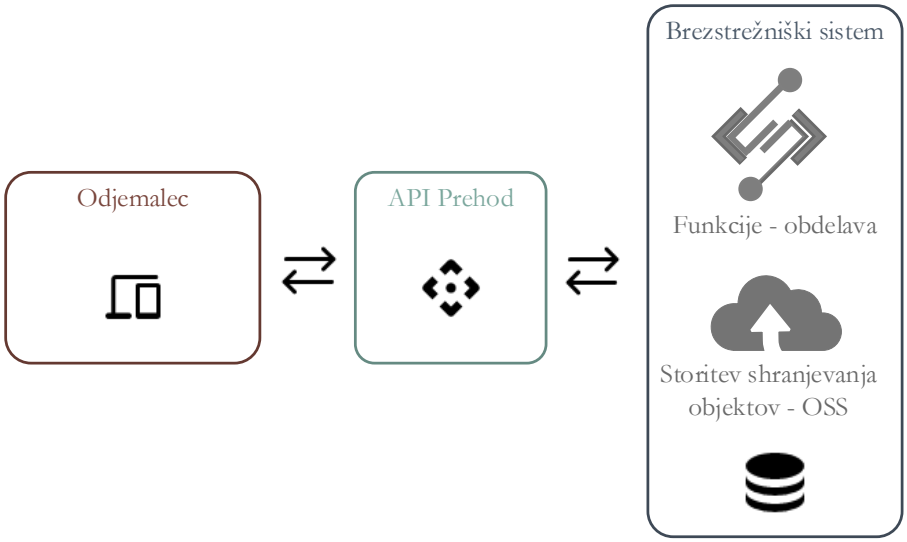
- mikrostoritvena arhitektura je stil, ki temelji na storitvah, ki za izpolnjevanje ciljev komunicirajo preko omrežja,
- komunikacija poteka preko sinhornih ali asinhornih protokolov,
- storitve so organizirane okrog organizacijskih zmogljivosti in omogočajo postopno implementacijo funkcionalnosti posameznega področja, neodvisno od drugih področij in storitev,
- vsaka mikrostoritev je avtonomna, zato obstaja možnost implementacije posameznih storitev v različnih programskih jezikih, platformah z različno strojno opremo ter različnih podatkovnih baz v ozadju,
- mikrostoritve so majhne, omogočajo pošiljanje sporočil, omejene so s kontekstom, v katerem delujejo,
- vsaka storitev deluje v svojem okolju, kar omogoča posodabljanje in namestitev novih različic z avtomatiziranimi procesi,
- vsaka storitev ima svoj načrt avtomatizacije (testiranje, posodobitve, prilagoditve, integriranje in vzdrževanje).

2.2. Brezstrežniško računalništvo

Računalništvo v oblaku je uporaba računalniške moči, pomnilnika, podatkovnih baz, storitev in preostalih IT virov na zahtevo. Praviloma se uporaba takšnih oblačnih storitev preko interneta zaračunava po modelu »plačevanje po uporabi« (angl. pay-as-you-go). Kot je splošno znano gre pri pojmu oblak za skupek računalnikov, ki se nahajajo na različnih lokacijah po svetu in so v lasti *ponudnikov oblačnih storitev*, kot so Amazon Web Service (AWS), Google Cloud in Microsoft Azure. Načelo računalništva v oblaku ostaja nespremenjeno a kljub temu doživlja določene spremembe. Ponudniki oblačnih storitev težijo k čim višji skalabilnosti, visoki odzivnosti storitev in agilnemu pristopu pri zagotavljanju virov, kar je zaradi vse hitrejšega razvoja na področju tehnologij za virtualizacijo tudi mogoče. V začetnih fazah razvoja oblačnega računalništva se je virtualizacija uporabljala kot sredstvo za konsolidacijo programske opreme in storitev, s katero je bila dosežena maksimalna izkoriščenost virov in enostavno upravljanje. V začetni fazi je prišlo do skupne souporabe strojne opreme. V naslednji fazi je bil na strežniku ustvarjen nabor virtualnih strojev (VM, angl. virtual machine), vsak VM je nosil kopijo operacijskega sistema. Razvoj se je nadaljeval z uporabo koncepta zabojnikov (angl. container). Zabojniki so platforma za shranjevanje virov, potrebnih za izvajanje določene aplikacije. V primerjavi z VM-ji, zabojniki omogočajo večjo abstrakcijo virov. Tako je zagotavljanje virov veliko hitrejše kot pri uporabi VM. Učinkovitost in hitro zagotavljanje virov s strani zabojnikov je omejena z osnovnimi infrastrukturnimi elementi, imenovanimi strežniki. Kot prikazuje slika 2 je brezstrežniško računalništvo model združevanja in uporabe virov, ki vključuje operacijski sistem, izvajalna okolja in strojno opremo. [5, 6, 8]

Pri brezstrežniškem računalništvu gre za uporabo dogodkovno vodenega razvoja informacijskih rešitev, ki torej temelji na dogodkih, kjer računalniške vire, potrebne za zagon kode, namesto nas upravlja ponudnik storitev v oblaku. Nanaša se na platforme, ki uporabnikom omogočajo izvajanje določenega dela kode na zahtevo. Imenuje se brezstrežniško (še vedno so v uporabi strežniki, gre za dodatno stopnjo abstrakcije), saj uporabnikom ni potrebno vzdrževati fizičnega ali virtualnega strežnika za izvajanje kode. Praviloma je potrebno v kateri koli organizaciji, kjer ne uporabljajo brestrežniških platform, vzdrževati in upravljati računalniške vire, ne glede na to, ali jih uporabljajo ali ne. Ob uporabi brezstrežniškega pristopa lahko svojo kodo gostimo in izvajamo v oddaljenem računalniškem viru, ki ga zagotovi ponudnik storitev, pri čemer moramo plačati za dejansko uporabo brez stroškov

vzdrževanja in upravljanja strežnika. Prav tako nismo sami odgovorni za morebitne nedejavnosti in izpade strežnikov. Brezstrežniško računalništvo ne odpravi uporabe strežnika, temveč le pomaga, da ga premaknemo v zaledje in s tem preložimo odgovornost na ponudnike teh storitev. Razvijalci se lahko osredotočijo na načrtovanje svoje informacijske rešitve. [6]



Slika 2: Brezstrežniška arhitektura.
Vir: lasten.

Ponudniki brezstrežniških tehnologij dinamično dodelijo strežnike, ki so na voljo uporabniku. Koda se ob klicu, ki ga sproži nek dogodek, izvaja v vsebnikih brez stanja. Izvajanje te kode lahko traja dalj časa in v več ponovitvah, za kar ni potreben ponoven klic. Poleg tega brezstrežniške rešitve vključujejo različne tehnologije, med katerimi sta dve ključnega pomena imenovani zaledje kot storitev (angl. Backend-as-a-service) ali BaaS in funkcije kot storitev ali krajše (angl. Functions-as-a-service) FaaS. *Baas* omogoča zamenjavo klasičnih strežniških komponent s standardnimi storitvami. Razvijalcem omogoča izkoriščanje zunanjih zmogljivosti (angl. outsourcing) v obliki zalednega dela, tako da se lahko razvijalci osredotočijo na pisanje in vzdrževanje logike uporabniških vmesnikov. Sistemi BaaS ponujajo storitve za avtentikacijo, upravljanje podatkovnih baz, shranjevanje v oblaku , gostovanje in podobno. FaaS je okolje, ki se uporablja za izvajanje programske opreme. Kot že omenjeno so brezstrežniške aplikacije dogodkovno vodeni sistemi, ki temeljijo na oblaku, kjer se razvoj aplikacij opira izključno na kombinacijo storitev tretjih oseb, logike na strani odjemalca in klicev oddaljenih postopkov, ki gostujejo v oblaku. FaaS razvijalcem omogoča uvedbo kode, ki se po sprožitvi izvede v izoliranem okolju. Vsaka funkcija običajno opisuje majhen del celotne aplikacije. Čas izvajanja funkcij je običajno omejen (npr. 15 min za AWS Lambda). Funkcije niso stalno aktivne, namesto tega jih platforme FaaS instancirajo po potrebi. Funkcije sprožijo dogodki, kot so zahteve odjemalcev, dogodki, ki jih ustvarijo zunanji sistemi, podatkovni tokovi ali drugi. Ponudnik FaaS sistemov so nato odgovorni za horizontalno skaliranje izvajanja funkcij v odvisnosti glede na število vhodnih dogodkov. [9]

Brezstrežniške aplikacije je mogoče razviti v več kontekstih, kljub temu so lahko na nekaterih področjih omejene. Na primer, dolgo delujoče funkcije, kot je izvajanje strojnega učenja ali dolgo delujoči algoritmi, imajo lahko težave s časovno omejitvijo. Takšne stalne delovne obremenitve lahko povzročijo višje stroške v primerjavi z neomejenim izvajanjem računalniških storitev na zahtevo, ki jih lahko najdemo pri postavitvi lastne mikrostoritvene arhitekture. [9, 10]

Omenjena pristopa BaaS in Faas sta primarna predstavnika implementacije brestrežniškega arhitekturnega stila. Predvsem model FaaS je po svojih karakteristikah (preprostosti in elastičnosti) popolnoma v skladu z načeli brestrežniškega računalništva. Omenjena modela lahko, kot je razvidno iz literature, pomotoma enačimo z modelom zabojnik kot storitev (angl. Container-as-a-Service) ali CaaS, ki je en izmed novejših pristopov v oblačnem računalništvu in spada med zbrane načrtovalske vzorce uvajanja storitev, pri uporabi mikrostoritvenega

arhitekturnega stila [4]. Glavni namen tega modela, je uporaba virtualizacije, ki temelji na zabojniških strukturah. V primerjavi z modelom FaaS, CaaS doda dodaten sloj v obliki zabojnikov, v katere zapakiramo bodisi uporabnikove aplikacije ali storitve. Zabojniki se namestijo na izbrano oblačno platformo, ki ob sprožitvenem dogodku zagotovi ali onemogoči posamezen zabojnik, odvisno od trenutnih zahtev. Dodaten nivo abstrakcije, v obliki pakiranja storitev v zabojnike, doda na kompleksnosti, a nam hkrati odpira dodatne možnosti na področju prilagajanja izvajalnega okolja in izvajanja aplikacij [11].

2.3. Razlike med arhitekturnima pristopoma

Kot smo že omenili ima brezstrežniško računalništvo več skupnih značilnosti z mikrostoritvami, vendar ima tudi nekaj razlik. V tabeli 1 prikazujemo primerjavo mikrostoritvene in brezstrežniške arhitekture po posameznih področjih, ki so pomemben faktor pri odločanju katero izmed arhitektur izbrati za razvoj informacijske rešitve.

Tabela 1: Prednosti in slabosti mikrostoritvene in brezstrežniške arhitekture.

	Mikrostoritvena arhitektura	Brezstrežniška arhitektura
Avtonomnost	Na nivoju posamezne mikrostoritve, ki deluje neodvisno od drugih in ima svoj življenjski cikel ter je zadolžena za določeno poslovno zmogljivost.	Na nivoju posamezne funkcije, ki bi morale biti oblikovane tako, da opravijo eno samo delo kot odgovor na dogodek.
Skalabilnost	Omogočeno je povečevanje ali zmanjšanje zmogljivosti storitev glede na potrebe. Lahko postane težavno, kadar za skaliranje skrbimo sami ali kršimo načela horizontalne skalabilnosti (npr. ohranjanje stanja).	Popolnoma samodejno skaliranje funkcij, ki se prilagajajo neodvisno od obremenitve. Zelo veliko zahtev se lahko obravnava kot zahteva enega odjemalca.
Odpornost	Vsaka mikrostoritev deluje samostojno v svojem okolju, tako izpad ene storitve ne pomeni izpad celotne aplikacije.	Ob izpadu posamezne funkcije ali ostalih delov smo odvisni od izbranega ponudnika, ki mora zagotoviti ustrezno politiko v primeru izpada.
Fleksibilnost	Vsako storitev lahko zgradimo z uporabo jezika ali ogrodja, ki ga potrebujemo, ne da bi to vplivalo na komunikacijo med mikrostoritvami.	Odvisni smo od izbranega ponudnika. Nekateri omogočajo hkratno podporo več jezikom in ogrodjem (npr. Google Cloud) – vsaka funkcija ima svoje izvajalno okolje – nekateri hkrati samo enemu jeziku (npr. Firebase).
Decentralizirano upravljanje podatkov	Mikrostoritve ponujajo možnost posameznega in neodvisnega dostopa do izvora podatkov, kar pomeni, da vsaka mikrostoritev upravlja s svojo podatkovno bazo.	Razvijalci lahko napišejo in uvedejo kodo, medtem ko ponudnik oblaka zagotovi strežnike za izvajanje njihovih aplikacij, podatkovne baze in sisteme za shranjevanje v poljubnem obsegu. Odvisni smo od izbranega ponudnika – praviloma izbrana podatkovna baza.

Vir: [4, 9, 11, 12]

Prednosti in slabosti posameznega arhitekturnega pristopa se pokažejo, ko želimo implementirati informacijsko rešitev, ki ponuja možnost uporabe tako mikrostoritvene, kot brezstrežniške arhitekture. V nadaljevanju bomo podali ključne iztočnice, ki vplivajo na izbiro arhitekturnega stila [11]:

– *Brezstrežniški sistemi lahko imajo ob začetni konfiguraciji podaljšanj čas prvega zagona.*

Kadarkoli je funkcija sprožena ali poklicana preko zahteve uporabnika, je razporejena v na novo ustvarjen vsebnik. Tako obstaja določeno kratko časovno obdobje, ko zahteva počaka, dokler ni vsebnik pripravljen za izvedbo. Ta pojav se v programerskem inženirstvu imenuje problem hladnega zagona.

– *Strategija uvajanja mikrostoritev ima težave z uravnoteženjem obremenitve in prerazporeditvijo prometa.*

Kljub morebiti daljšemu prvemu zagonu, so po začetnem obdobju brezstrežniški sistemi zelo stabilni. Nasprotno imajo lahko mikrostoritvene arhitekture težave ob povečani obremenitvi. Zaradi potrebe po prerazporeditvi prometa po dodatnih instancah storitev in hitrem skaliranju ob višji obremenitvi, se soočamo z zakasnitvijo pri podajanju odgovorov na zahteve. Zaradi narave brezstrežniških arhitektur so ti sistemi v takšnih situacijah praviloma hitrejši in ponujajo enakomerno časovno zakasnitev preko celotnega obdobja delovanja.

– *Strategija uvajanja mikrostoritev je uspešnejša pri pridobivanju majhnih in ponavljajočih se zahtev.*

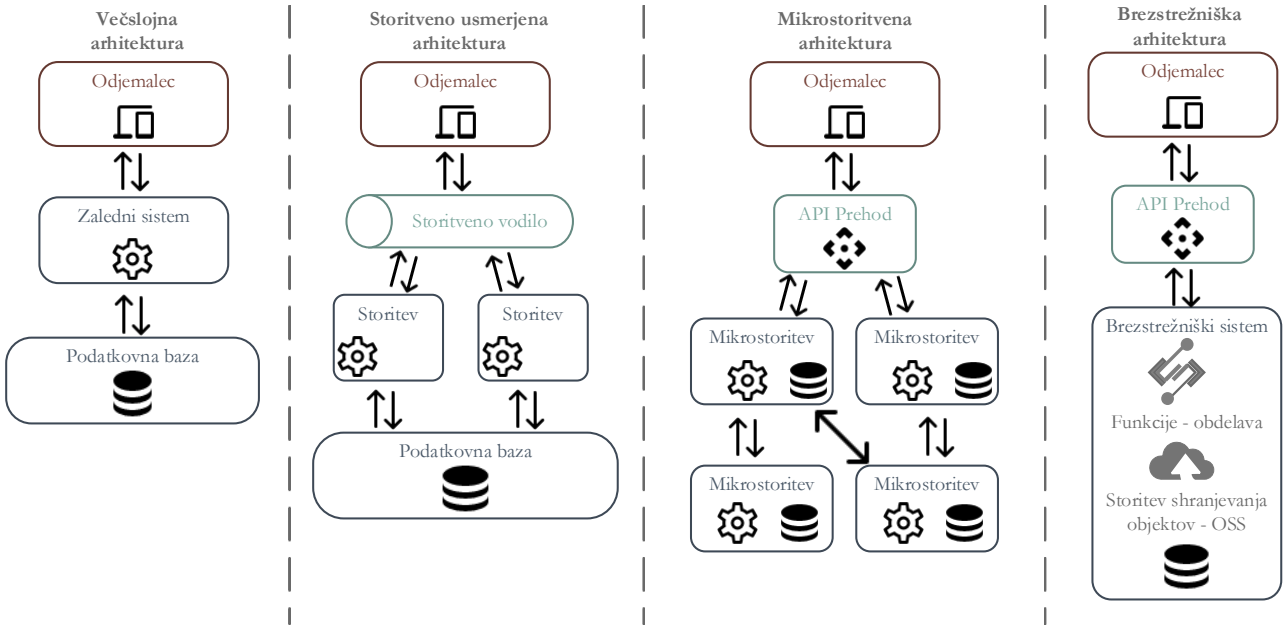
Kadar vemo, da bo imel naš informacijski sistem veliko število preprostih ponavljajočih zahtev z majhno velikostjo, je ustrezneje uporabiti mikrostoritveno arhitekturo. Izognemo se lahko velikemu strošku, ki bi nastal, kadar bi nam ponudnik brezstrežniške platforme zaračunaval za obdelavo vseh teh majhnih zahtev. Prav tako nimamo težav s hladnim zagonom, ki je najbolj opazen ob proženju majhnih zahtev.

– *Brezstrežniški sistemi so okretnejši in omogočajo lažjo konfiguriranje, ko govorimo o skalabilnosti.*

Ena izmed glavnih prednosti brezstrežniških sistemov, je odlična podpora samodejni skalabilnosti za katero skrbi infrastruktura izbranega ponudnika. Pri uporabi mikrostoritvene arhitekture je to pomanjkljivost mogoče odpraviti s konfiguracijo ustreznega mehanizma, po navadi z uporabo ustreznega orkestratorja (npr. Kubernetes).

3 Evolucija arhitektur programske opreme

Mikrostoritve izvirajo iz razvoja programske arhitekture z namenom zmanjšanja kompleksnosti monolitnih in storitveno usmerjenih sistemov. Pri mikrostoritveni arhitekturi gre za majhne in avtonomne storitve, ki imajo en sam jasno določen namen. Omogočajo navpično razgradnjo aplikacij v podmnožico poslovno usmerjenih neodvisnih storitev, povezanih z lahkimi komunikacijskimi protokoli ali vmesniki (API). Vsako storitev lahko neodvisno razvijejo, uvedejo in preizkusijo različne razvojne skupine z uporabo različnih tehnoloških skladov. Mikrostoritve je mogoče razviti v različnih programskih jezikih, lahko se prilagajajo neodvisno od drugih storitev. Namestiti jih je mogoče na strojno opremo, ki najbolj ustreza njihovim potrebam. Običajno so odzivne na njihov vmesnik, ki je primarni mehanizem za interakcijo z logiko. Slika 3 prikazuje razvoj arhitekturnih pristopov informacijskih sistemov.



Slika 3: Evolucija arhitektur.
Vir: lasten.

Nasprotno se brezstrežniške tehnologije odzivajo na dogodke, medtem ko so API-ji predvsem mehanizmi, ki so namenjeni za generiranje dogodkov. Sprožitev funkcije lahko izvedemo iz različnih virov dogodkov, kot so podatkovna baza ali sporočilni posrednik, pri čemer ne potrebujemo vmesnika API. Ključna razlika med mikrostoritvami in brezstrežniškimi sistemi je manjša razdrobljenost brezstrežniških funkcij in sprejetje paradigme, ki temelji na dogodkih. Kljub temu, da je mogoče razviti tudi mikrostoritve, ki so vodene na podlagi dogodkov, je za dogodkovno vodene sisteme zelo priporočljivo uporabiti brezstrežniški pristop. Ena izmed možnosti združevanja mikrostoritvene in brezstrežniške arhitekture je z uporabo različnih vzorcev**.** Kot primer lahko navedemo uporabo vzorca Saga, vratar (angl. Gatekeeper) in podobni. [9]

3.1. Razlikovanje mikrostoritvenega in brezstrežniškega arhitekturnega pristopa

Čeprav brezstrežniški arhitekturni pristop še morda ni dozorel, v primerjavi z mikrostoritvami ne izgublja v smislu svoje izvedljivosti, zanesljivosti in potenciala v prihodnosti. Brezstrežniška arhitektura ni neposredno povezana z mikrostoritvami, vendar obstajajo podobnosti med obema, kot so delitev poslovanja, možnost razvoja aplikacij brez stanja in agilne funkcionalnosti. [12]

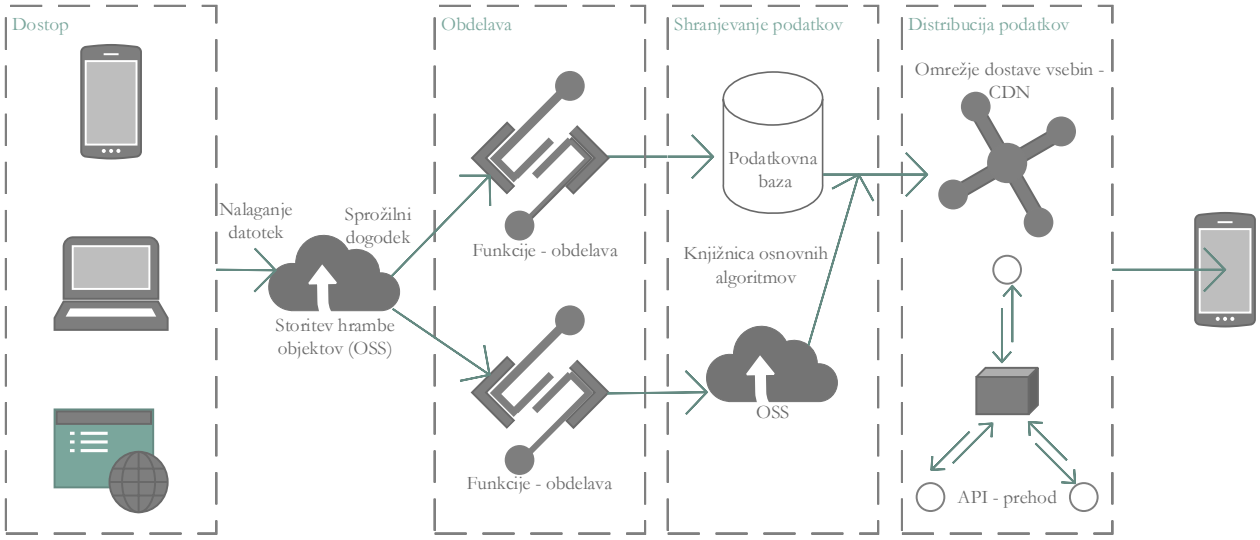
Pri razvoju novih sistemov v programskem inženirstvu, sledimo uporabi dokazano uspešnih gradnikov in izkušenj, kar je eden izmed temeljev vsake inženirske discipline. Na višjem nivoju abstrakcije se srečamo s ponovno uporabo s pomočjo vzorcev (angl. software patterns). Ena izmed splošno sprejetih skupin načrtovalskih vzorcev mikrostoritvene arhitekture, ki nudi povezovanje uporabe mikrostoritvene in brezstrežniške arhitekture, je skupina imenovana Izvajanje mikrostoritev (angl. Deployment patterns), v kateri najdemo vzorce, ki so povezani z izvajanjem posameznih mikrostoritev in celotne informacijske rešitve. Vzorec, ki mu bomo posvetili pozornost, je vzorec imenovan uvedba brez strežnika.

– *Uvedba brez strežnika* (angl. Serverless deployment).

  Vzorec govori o tem, da je najprej potrebno izbrati ustrezno uvedbeno infrastrukturo, ki ne razkriva kateri strežniški koncept je uporabljen – fizični ali virtualni gostitelj, zabojniki. Torej izberemo ponudnika brezstrežniških storitev, ki ustreza našim zahtevam. Ponudnik nam zaračunava na podlagi uporabe njihove uvedbene infrastrukture in je zadolžen za upravljanje infrastrukture na nizki ravni. Na uvedbeno

infrastrukturo namestimo informacijsko rešitev, ki je za to primerna [4]. Opisan vzorec je prikazan na sliki 4.

Čeprav gre za vzorec, ki je naveden znotraj katalogov dobrih praks razvoja mikrostoritev, ne smemo pozabiti, da se pri uporabi tega vzorca oddaljujemo od uporabe mikrostoritvene arhitekture (in uporabimo brezstrežniki arhitekturni pristop). Vzorec uvedba brez strežnika nam omogoča komplementarno dopolnjevanje in možnost, da razmislimo kateri izmed arhitekturnih pristopov je ustreznejši za našo informacijsko rešitev.



Slika 4: Načrtovalski vzorec uvedba brez strežnika.
Vir: [4]

Ko so na trg prišle prve platforme brez strežnikov, so strokovnjaki pogosto začeli napačno graditi mikrostoritve z uporabo funkcij brez strežnikov. Na primer, informacijsko rešitev lahko zgradimo kot mikrostoritev s prehodom (API) - vmesnik storitve, za njim pa uporabimo brezstrežniške funkcije in logiko, ki deluje kot usmerjevalnik. Ta zelo preprost pristop omogoča aplikacijam neodvisno prilagajanje velikosti, vendar še vedno ne izkorišča pravilno prednosti brezstrežniškega računalništva. [9]

Mikrostoritve so najbolj primerne za dolgotrajne, kompleksne aplikacije, ki imajo velike zahteve glede virov in upravljanja. Funkcije brez strežnika vodijo dogodki in se zato izvajajo le, ko je to potrebno. Ko je izvedba končana, se računalniška instanca, ki izvaja funkcijo, prekine. Brezstrežniške tehnologije so bolj primerne za obdelavo dogodkov in naloge, ki ne zahtevajo veliko virov. Brezstrežniški sistemi se prilagajajo samodejno, ne da bi za to potrebovali posebno infrastrukturo in so zato primernejši, ko razvijalci potrebujejo samodejno prilagajanje in nižje stroške izvajanja. Na drugi strani nam mikrostoritve ponujano večjo prilagodljivost, saj nismo odvisni od tretjih ponudnikov [12]. Povzetek osnovnih značilnosti posameznega arhitekturnega pristopa je zbran v tabeli 2.

Tabela 2: Karakteristike mikrostoritvene in brezstrežniške arhitekture.

	Mikrostoritvena arhitektura	Brezstrežniška arhitektura
Poslovni razcep	Vezano na storitev	Vezano na funkcijo
Zahteva strežnika	Deljenje stanja pomnilnika med klici	Popolnoma brez stanja
Tretje odvisnosti	Prosta izbira tujih odvisnosti	Znašanje na tuje odvisnosti - zagotavlja BaaS
Nivo abstrakcije	Razvojni model	Računalniška platforma

Vir: [12]

4 Zaključek

Na podlagi pregleda in opisa mikrostoritvene in brezstrežniške arhitekture ugotavljamo, da nobena posamezna vrsta arhitekturnega stila ne more ustrezati vsem vrstam aplikacij. Iz razprave v tem delu in po analizi brezstrežniških sistemov ter mikrostoritev ugotavljamo, da ima vsaka tehnologija svoje prednosti in slabosti. Vsak izmed obeh pregledanih arhitekturnih stilov, rešuje izzive na svoj način. Tako moramo biti pazljivi, da ju ustrezno ločujemo, saj se razlikujeta v svojih osnovnih načelih. Odvisno od zahtev sistema, lahko gresta oba arhitekturna stila z roko v roki, pri čemer ne smemo kršiti pravil implementacije. Primer sočasne uporabe je najpreprosteje ponazoriti s primerom informacijske rešitve, ki ima uporabniški vmesnik zgrajen s pomočjo več mikrostoritev, od katerih se pričakuje, da se nenehno izvajajo. Modele brezstrežniškega arhitekturnega stila pa lahko uporabimo, kot dopolnitve k celotnemu sitemu, ki skrbijo za proženje občasnih časovnih in dogodkovnih funkcij ali za varnost celotnega sistema. Kljub temu, da gre pri brezstrežniškem arhitekurnem stilu za novejši pristop, ki vedno bolj pridobiva na veljavi, se izogibamo nesmiselnemu preoblikovanju sistemov, ki uporabljajo mikrostoritveni arhitekturni stil na način, da bi vsako posamezno mikrostoritev poskušali implementirati v obliki več funkcij, ki se izvajajo na brezstrežniškem sistemu. Omeniti je potrebno, da brezstrežniški arhitekurni stil predstavlja enega izmed arhitekturnih stilov in ni namenjen zamenjavi za dalj časa uveljavljen mikrostoritveni arhitekturni stil. Na podlagi primerov uporabe in poslovnih potreb lahko uporabniki skrbno prepoznajo, kateri arhitekturni stil bodo uporabili in ga ustrezno uvedejo glede na funkcionalne potrebe. Brezstrežniška arhitektura je primerna za dogodkovno vodene sisteme, kjer se mora izvesti dejanje, ko pride do sproženega dogodka. Večinoma gre za funkcijsko voden pristop, kjer je nadzor pri gostitelju in ne pri uporabniku. Osrednja modela, ki se pojavljata pri brezstrežniškem arhitekturnem stilu sta BaaS in FaaS. Posebej moramo opozoriti na model CaaS, ki ponazarja enega izmed načinov izvajanja mikrostoritev pri uporabi mikrostoritvenega arhitekturnega stila in se oddaljuje od načel in modelov, ki so značilni za sisteme, ki uporabljajo brezstrežniški arhitekturni stil. Mikrostoritve so modularne, domenske, samostojne skupine storitev, ki lahko delujejo neodvisno druga od druge. Oba arhitekturna stila nam pri gradnji sodobnih rešitev v oblaku in s pravilno izbiro pristopov omogočata, da zgradimo zelo robustne spletne aplikacije z minimalnimi stroški, ki so ob enem zelo prožne, razširljive ter varne.

Literatura

[1] V. Raj and S. Ravichandra, "Microservices: A perfect SOA based solution for Enterprise Applications compared to Web Services," in *3rd IEEE International Conference on Recent Trends in Electronics, Information & Communication Technology (RTEICT)*, 2018, p. 1531–1536.

[2] J. Lewis in M. Fowler, „Microservices a definition of this new architectural term," [Elektronski]. Na voljo: https://martinfowler.com/articles/microservices.html [dosegljivo 7/2022].

[3] N. Alshuqayran, N. Ali and R. Evans, "Systematic Mapping Study in Microservice Architecture«," in *IEEE 9th International Conference on Service-Oriented Computing and Applications (SOCA)*, 2016, p. 44–51.

[4] T. Hliš and L. Pavlič, *Dobre prakse pri razvoju mikrostoritev : študijski program: Informatika in tehnologije komuniciranja,* 2021.

[5] E. Eyk, Iosup, &. Alexandru, J. Grohmann, Eismann, &. Simon, Bauer, &. André, L. Versluis, L. Toader, N. Schmitt, Herbst and C. Nikolas & Abad, "The SPEC-RG Reference Architecture for FaaS: From Microservices and Containers to Serverless Platforms," *IEEE Internet Computing. PP,* p. 1–1, 2019.

[6] B. Jambunathan and K. Yoganathan, "Architecture Decision on using Microservices or Serverless Functions with Containers," in *2018 International Conference on Current Trends towards Converging Technologies (ICCTCT)*, 2018, p. 1–7.

[7] R. Collier, O'neill, &. Eoin, Lillis, &. O. David and G. Hare, *MAMS: Multi-Agent MicroServices,* 2019.

[8] R. Rajan, "Serverless Architecture - A Revolution in Cloud Computing," in *Tenth International Conference on Advanced Computing (ICoAC)*, 2018, p. 88–93.

[9] D. Taibi, J. Spillner and K. Wawruch, "Serverless Computing-Where Are We Now, and Where Are We Heading?," in *IEEE Software*, vol. 38, 2021, p. 25–31.

[10] A. Rajan, *A review on serverless architectures - function as a service (FaaS) in cloud computing. TELKOMNIKA (Telecommunication Computing Electronics and Control),* 2020.

[11] K. Burkat, M. Pawlik, B. Balis, M. Malawski, K. Vahi, M. Rynge, R. F. d. Silva in E. Deelman, „Serverless Containers – Rising Viable Approach to Scientific Workflows," v *2021 IEEE 17th International Conference on eScience (eScience)*, 2021.

[12] C.-F. &. Fan, Jindal and M. Gerndt, *Microservices vs Serverless: A Performance Comparison on a Cloud-native Web Application,* 2020.

[13] L. Jiang, Pei and J. Zhao, "Overview Of Serverless Architecture Research," in *Journal of Physics: Conference Series*, 2020.