Univerzitet u Beogradu

Matematički fakultet

Master rad

Razvoj REST servisa u stilu arhitekture

“bez servera” na platformi Microsoft Azure

*Student: Mentor:*

Miloš Milovanović dr Vladimir Filipović

*Mentor*: prof. dr Vladimir Filipović *Matematički fakultet, Univerzitet u Beogradu   
  
  
  
Članovi komisije*: prof. dr Saša Malkov  
 *Matematički fakultet, Univerzitet u Beogradu* dr Aleksandar Kartelj  
 *Matematički fakultet, Univerzitet u Beogradu*

***Datum odbrane*: \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

Apstrakt

Sadržaj

[1 Uvod 5](#_Toc72958646)

[2 Nastanak računarstva “bez servera” 7](#_Toc72958647)

[2.1 Monolitna, slojevita i mikroservisna arhitektura 7](#_Toc72958648)

[2.2 Virtuelizacija zasnovana na kontejnerima 9](#_Toc72958649)

[2.3 Računarstvo u oblaku, modeli servisa 10](#_Toc72958650)

[2.4 Računarstvo “bez servera” 12](#_Toc72958651)

[2.4.1 Funkcija kao servis 13](#_Toc72958652)

[2.4.2 Osobine 14](#_Toc72958653)

[2.4.3 Prednosti i nedostaci 15](#_Toc72958654)

[3 Platforme na modelu funkcija kao servis 17](#_Toc72958655)

[3.1 Pregled 17](#_Toc72958656)

[3.2 Azure Functions 18](#_Toc72958657)

[3.2.1 Izvršno okruženje i planovi korišćenja 18](#_Toc72958658)

[3.2.2 Funkcije i aplikacije funkcija 19](#_Toc72958659)

[3.2.3 Lokalni razvoj 20](#_Toc72958660)

[3.2.4 Okidači i vezivanja 22](#_Toc72958661)

[3.2.5 Trajne funkcije 24](#_Toc72958662)

[3.2.6 Postavljanje na Azure platformu 25](#_Toc72958663)

[4 Razvoj REST servisa Recepti API 27](#_Toc72958664)

[4.1 Funkcionalni opis i arhitektura 27](#_Toc72958665)

[4.2 Implementacija servisa 27](#_Toc72958666)

[4.2.1 Struktura projekta 27](#_Toc72958667)

[4.2.2 Baza podataka 27](#_Toc72958668)

[4.2.3 API resursi 27](#_Toc72958669)

[4.2.4 Implementacija funkcija 27](#_Toc72958670)

[5 Gradjenje, hostovanje i testiranje servisa Recepti API 28](#_Toc72958671)

[5.1 Resursi na platformi u oblaku i postavljanje servisa 28](#_Toc72958672)

[5.2 Testiranje 28](#_Toc72958673)

[5.3 Nadgledanje i održavanje 28](#_Toc72958674)

[6 Zaključak 29](#_Toc72958675)

1 Uvod

Isporučioci javnih platformi za računarstvo u oblaku (eng. Public cloud vendors) beleže stalni rast u korišćenju svojih usluga od njihovog nastanka. Poslednjih godina primetno je da je ovaj rast mnogostruko uvećan. Celokupno tržište servisa računarstva u oblaku poraslo je za 17.5% u toku 2019. godine u odnosu na prethodnu po istraživanju kompanije Gartner [1]. Sa porastom popularnosti i dostupnosti ovih servisa, mnoge kompanije odlučuju se da hostovanje svog softvera delom ili u potpunosti premeste sa interne infrastrukture na neke od platformi u oblaku.

Računarstvo u oblaku u svojim počecima predstavljalo je iznajmljivanje infrastrukturnih resursa kao što su serveri, mrežna infrastruktura ili sistemi za skladištenje podataka, tako da su korisniku dostupni na zahtev i naplativi po utrošku ili vremenskoj rezervisanosti. Razvijaocu je bilo omogućeno da upravlja, održava i nadgleda svoju infrastrukturu u oblaku putem servisa. Razvojem koncepata kao što su kontejneri (eng. Containers) i njihova orkestracija stvorili su se uslovi da se upravljanje infrastrukturnim resursima može obavljati na apstraktniji i automatizovan način. Sa tim dolazi i do stvaranja potpuno novog načina izvršavanja na platformama u oblaku pod nazivom funkcija kao servis (eng. Function as a Service). U ovom slučaju razvijaoc je zadužen samo za aplikativni kod koji organizovan u obliku funkcija, dok se o infrastrukturnom okruženju za njihovo izvršavanje stara sama platforma.

Na taj način stvorili su se uslovi za pojavu takozvane arhitekture “bez servera" (eng. Serverless arhitecture), kao novog arhitekturnog obrasca za projektovanje i razvoj aplikacija i servisa u oblaku. Cilj ovog rada je da prikaže njen smisao i karakteristike, kao i pregled funkcionalnosti i praktična primena Azure funkcija (eng. Azure Functions) kao jedne od tri najpopularnije platforme ovog tipa.

U drugom poglavlju rada biće opisana istorija i razvoj arhitekture, od prvih oblika softverske arhitekture, kroz pojavu računarstva u oblaku do arhitekturnih koncepata koji su uticali na njeno stvaranje u današnjem obliku. Biće objašnjeni teorijski aspekti arhitekture “bez servera" i modela izvršavanja funkcija kao servis, njihove glavne prednosti i nedostaci.

U trećem poglavlju biće dat kratak prikaz Micosoft Azure platforme. Pored toga će detaljno biti opisane Azure funkcije od načina razvoja funkcija, njihovih tipova, okidača i vezivanja, do internog načina izvršavanja i naplate.

U četvrtom poglavlju će biti demonstrirana implementacija REST API servisa korišćenjem arhitekture i tehnika opisanih u prethodna dva poglavlja. Servis koji će biti prikazan zamišljen je kao deo Recepti API veb servisa za kulinarske recepte. Biće dati funkcionalni opis, shema baze podataka i značajni delovi koda, a celokupan kod servisa biće dostupan javno na adresi <https://github.com/milosmi11166/Master>.

U petom poglavlju biće prikazano hostovanje, podešavanje kontinualne integracije, testiranje i nadgledanje servisa Recepti API na Microsoft Azure platformi.

U poslednjem šestom poglavlju biće izveden zaključak.

2 Nastanak računarstva “bez servera”

U ovom poglavlju biće predstavljen razvoj računarstva “bez servera” i tehnologije i arhitekture koje su na njega uticale. Biće dat prikaz mikroservisne arhitekture, virtualizacije zasnovane na kontejnerima i modela servisa računarstva u oblaku. Biće dati koncepti računarstva “bez servera” i modela funkcija kao servis, njihove karakteristike i prednosti i nedostaci.

2.1 Monolitna, slojevita i mikroservisna arhitektura

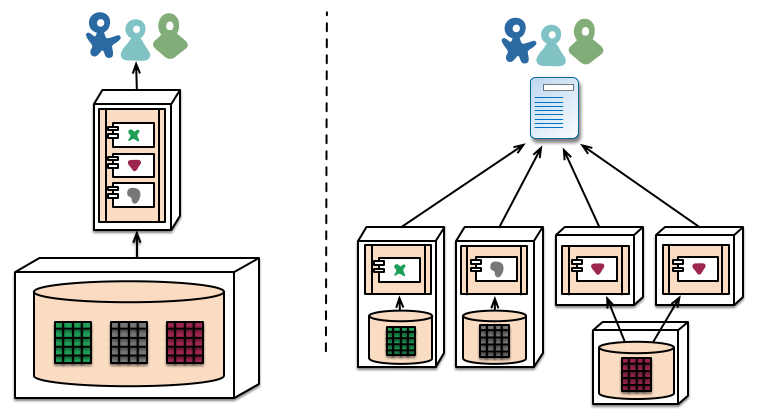
U domenu veb aplikacija, monolitnim aplikacijama nazivamo jednoslojne aplikacije koje u sebi sadrže implementaciju poslovne logike, pristupa sistemima za čuvanje podataka i korisničkog interfejsa. U slučaju kada su ovi elementi jasno odvojeni koristi se i naziv slojevita arhitektura (eng. N-Tier architecture). Aplikacije se sastoje se iz jedne ili nekoliko usko povezanih izvršnih datoteka koje se najčešće postavljaju na virtualnim mašinama ili direktno na operativnom sistemu serverskog računara. U zavisnosti od veličine i kompleksnosti razvoj je organizovan tako da jedan ili više razvojnih timova rade na zajedničkom repozitorijumu sa kodom aplikacije.

U slučaju manjih ili manje složenih aplikacija slojevita arhitektura ima dosta predosti zbog svoje jednostavnosti za razvoj, testiranje i postavljanje na server. Kod većih i kompleksnijih aplikacija pojavljuju se neki od nedostataka ovog pristupa. Sa većim brojem funkcionalnosti povećava se i složenost aplikacije što otežava izvodjenje izmena. Vremenom održavanje i odgovaranje na nove funkcionalne zahteve postaje dugotrajnije. Takodje greške pri izvršavanju na nekom delu aplikacije ili nekoj od funkcionalnosti mogu uticati na dostupnost cele aplikacije. Horizontalno skaliranje moguće je jedino postavljanjem više identičnih instanci cele aplikacije iza rasporedjivača opterećenja (eng. Load balancer).

Mikroservisna arhitektura je nastala sa ciljem da poveća fleksibilnost i proširivost aplikacije podelom na male labavo vezane servise koji se mogu nezavisno razvijati, testirati i postavljati na server. Servisi su dizajnirani tako da implementiraju minimalni broj operacija koje podržavaju jednu poslovnu funkcionalnost i imaju mogućnost medjusobnog komuniciranja sa drugim servisima. Ova ideja nije nova i mnogi koncepti zasnivaju se na starijoj servisno orijentisanoj arhitekturi (SOA).

Implementacija komunikacije zasniva se uglavnom na univerzalno poznatim komunikacionim stilovima kao što su REST (eng. Representational state transfer) ili gRPC (Remote procedure call) koji nisu zavisni od konkretne tehnologije razvoja. Na ovaj način omogućava se decentralizovana komunikacija i upravljanje servisima, za razliku od SOA kod koje su ovi procesi centralizovani i najčešće ih obavlja nezavisna komponenta. Proces razvoja servisa može biti podeljen na više repozitorijuma na sistemima za verzionisanje koda, mogu se koristiti različite tehnologije i programski jezici, kao i sistemi za skladištenje podataka. Takođe, servisi bi trebali da budu dizajnirani tako da su tolerantni na nedostupnost i kvarove, odnosno drugi servisi koji ih koriste moraju biti spremni na takve situacije i odgovoriti elegantno.

Servisi se mogu potpuno nezavisno postavljati na server i skalirati kao deo sistema, nasuprot monolitnim aplikacijama kod kojih je to moguće samo u celini. Postavljanju se na virtualnim mašinama servera, a često i koristeći neke od tehnologija za kontejnerizaciju. Sam proces postavljanja odvija se automatizovano kroz neki oblik kontinualne integracije, a pojavom računarstva u oblaku omogućeno je i automatizovano upravljanjem infrastrukturnim resursima koji su potrebni za pokretanje i rad servisa [2]. Sve ovo povećava kompleksnost operacionih zadataka prilikom razvoja, postavljanja na server i nadgledanja mikroservisnih aplikacija, što je i jedan od nedostataka ovog pristupa.

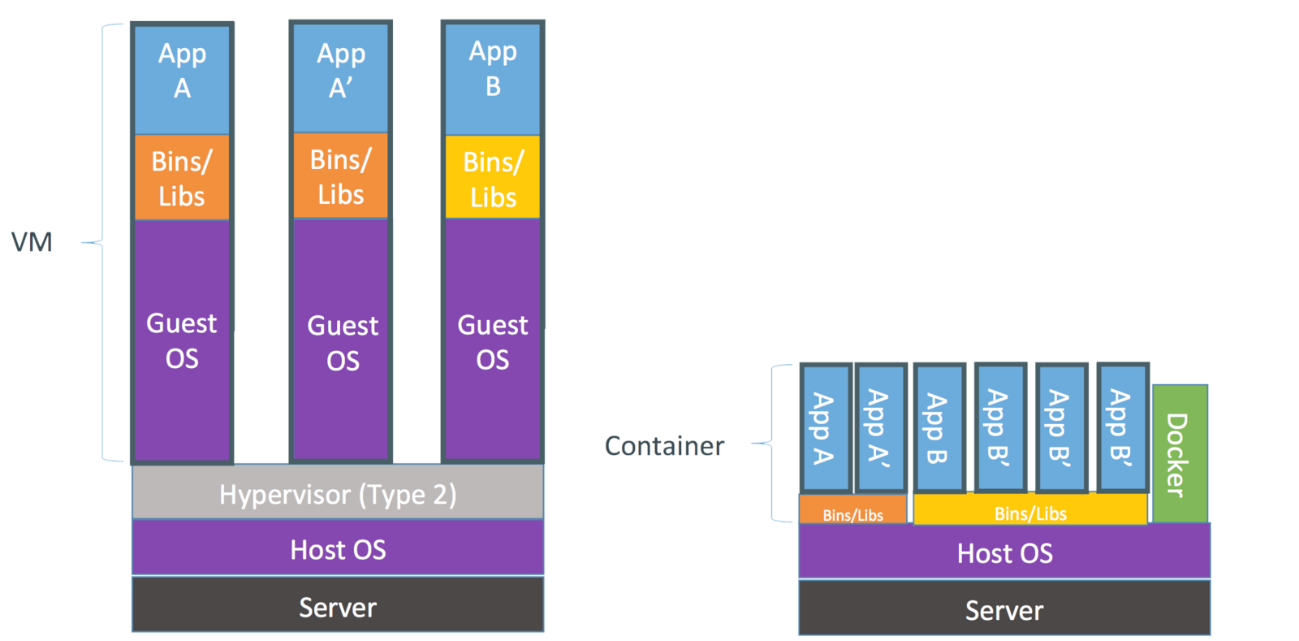


Slika 1 Slojevita i mikroservisna arhitektura [2]

2.2 Virtuelizacija zasnovana na kontejnerima

Softverski kontejneri su jedan od oblika virtualizacije na nivou operativnog sistema. U idealnom slučaju slika kontejnera (eng. Container image) enkapsulira jedan ili više procesa zajedno sa njihovim kodom, zavisnim bibliotekama, parametrima okruženja (eng. Environment variables) i drugim datotekama koje su potrebne prilikom njihovog izvršavanja. Kontejnerom nazivamo jednu insancu slike koju je moguće nezavisno izvršavati. U poređenju sa virtualnim mašinama koje pokreće hipervizor (eng. VM Hypervisor) i u sebi sadrže kompletan gostujući operativni sistem, softverski kontejneri koriste kernel operativnog sistema domaćina. Na Linuks operativnim sistemima izolacija izmedju kontejnera postiže se korišćenjem prostora imena (eng. Linux namespaces). Na ovaj način resursi kao što su pristup sistemu datoteka, alocirana memorija i mrežni portovi jednog kontejnera su izolovani od drugih. Na slici 2 prikazan je odnos kontejnera i virtualne mašine.

Jedna od najkorišćenijih platformi rad sa kontejnerima je Docker, sa repozitorijumom DockerHub na kome su javno dostupne mnogobrojne slike kontejnera kreirane od strane korisnika. Nove slike kontejnera definišu se preko komandi u posebnoj datoteci sa nazivom Dockerfile. Za pokretanje i upravljanje aplikacijama koje se sastoje od više kontejnera može se koristiti alat Docker Compose, sa konfiguracionim datotekama pisanim u YAML jeziku.



Slika 2 Odnos kontejnera i virtualne mašine

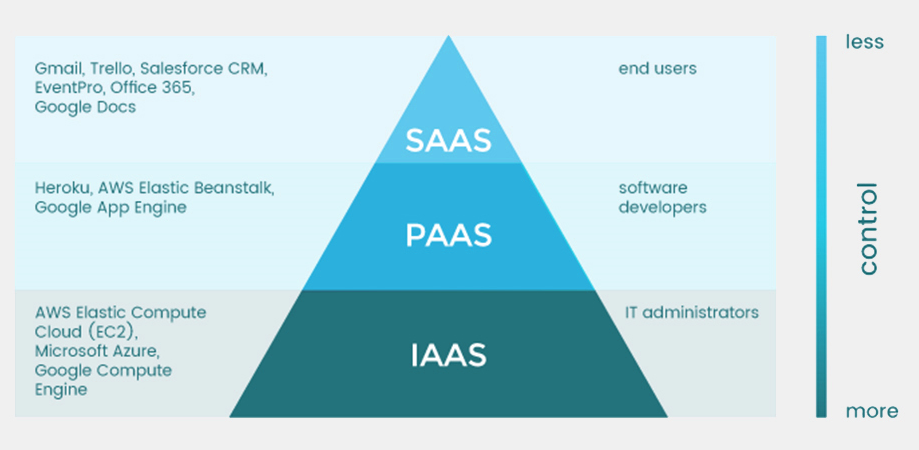
Orkestracija kontejnera je naziv za automatizovanu konfiguraciju, postavljanje, upravljanje i skaliranje sistema koji su zasnovani na kontejnerima. Kroz alate za orkestraciju moguće je definisati upotrebu serverskih resursa od strane kontejnera, komunikaciju izmedju samih kontejnera kao i sa eksternim činiocima. Pored toga omogućavaju pokretanje odnosno zaustavljanje instanci kontejnera kako bi se osiguralo željeno stanje aplikacije i nivo skaliranja. Neki od sistema ovog tipa su Kubernetes, Docker Swarm i drugi.

2.3 Računarstvo u oblaku, modeli servisa

asdasdasdasd

Prema nivou raspodele odgovornosti prilikom upravljanja resursima izmedju isporučioca platforme i korisnika, mogu se uočiti različiti modeli servisa. Tri glavna modela servisa računarstva u oblaku su opisana u nastavku.

* Infrastruktura kao servis (eng. Infrastructure as a service) se odnosi na rezervisanje korišćenja tri vrste infrastrukturnih resursa putem servisa. To su serveri, resursi za skladištenje podataka i mrežna infrastruktura. Jedinica rezervisanja servera najčešće je virtualna mašina sa željenim operativnim sistemom i definisanim hardverskim resursima za centralno procesiranje, memoriju i drugima. Kod servisa za skladištenje podataka moguće je rezervisati skladištenje generičkih objekata (eng. Blob Objects) ili specijalizovanih sistema za blokovsko skladištenje i skladištenje datoteka. Mrežni servisi omogućavaju povezivanje rezervisanih resursa u virtualnu mrežu na platformi u oblaku kao i njihovo povezivanje sa drugim mrežama ili Internetom. Plaćanje kod servisa koji pripadaju ovom modelu obavlja se najčešće po vremenu rezervisanosti, ali se može i razlikovati u zavisnosti od tipa servisa. Isporučioci na platformi uglavnom imaju u ponudi i servise koji olakšavaju operacione aktivnosti na rezervisanim infrastururnim resursima, preko servisa za nadgledanje, upravljanje log datotekama, pravljenja rezervnih kopija (eng. Backup), bezbednosti, oporavka od katastrofe (eng. Disaster recovery) i drugih.
* Platforma kao servis (eng. Platform as a service) se odnosi na rezervisanje korišćenja platforme za razvoj, postavljanje i izvršavanje aplikacija na platformi u oblaku. Ovakve platforme nude mogućnost izbora programskih jezika, razvojnih okvira, biblioteka, korišćenja kontejnera za virtualizaciju, kontinualne integracije i drugih alata prilikom razvoja aplikacija. Osim toga omogućavaju podešavanja platforme prilikom izvršavanja aplikacije korišćenjem parametara okruženja, nadgledanje, upravljanje log datotekama i podešavanja bezbednosti na nivou aplikacije. Razvijaoc nema mogućnost direktne kontrole i operacionih aktivnosti nad infrastrukturnim resursima koji se nalaze ispod platforme. Naplata kod ovakvih servisa može se razlikovati od vremena korišćenja, broja korisnika, količine prostora za skladištenje, broja zahteva odnosno transakcija i drugih.
* Softver kao servis (eng. Software as a service) predstavlja iznajmljivanje korišćenja gotovih aplikacija razvijenih od strane isporučioca ili trećih kompanija koje se izvršavaju na platformi u oblaku. Ove aplikacije najčešće se koriste preko korisničkog interfesa, dostupnog preko veb pregledača, mobilnih uređaja ili drugih. Korisnik nema kontrolu nad okruženjem aplikacije kao ni infrastrukturnim resursima prilikom njenog izvršavanja. Upravljanje je moguće jedino u samoj aplikaciji i to nad dodeljenim korisničkim nalozima i dozvolama, odnosno limitiranim skupom funkcionalnosti same aplikacije. Plaćanje kod ovog modela se često definiše pretplatom na mesečnom ili godišnjem nivou.



Slika 3. Modeli servisa računarstva u oblaku

2.4 Računarstvo “bez servera”

Iako izraz računarstvo “bez servera” (eng. Serverless computing) nagoveštava da ne postoji serverska komponenta u smislu hardvera i serverskih procesa kao dela arhitekture sistema, to nije u potpunosti tačno. Zapravo odnosi se na prebacivanje odgovornosti za upravljanje serverima i drugim resursima potrebnim za izvršavanje koda na treće lice. U velikom broju primera je u pitanju isporučioc platforme u oblaku, ali to ne mora uvek biti slučaj.

U osnovi računarstvo “bez servera” se može podeliti u dve grupe servisa. Prva grupa je takozvani zadnji kraj kao servis (eng. Back end as a service). To su servisi koji su u potpunosti razvijeni od strane trećih lica i hostovani na nekoj od platformi u oblaku. Kao takve moguće ih je integrisati u veb ili mobilnu aplikaciju razvijaoca. Ovde spadaju servisi različitih namena od autentikacije korisnika (Auth0, Okta), preko baza podataka (AWS Aurora, Firebase) do servisa za slanje notifikacija i mnogih drugih. Drugu i značajniju grupu čini model funkcija kao servis (eng. Function as a service) i on će biti detaljnije opisan u narednim sekcijama.

2.4.1 Funkcija kao servis

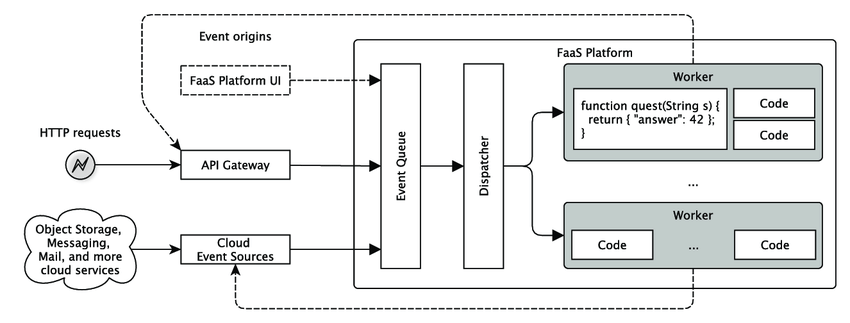
Funkcija kao servis je relativno nov model servisa računarstva u oblaku koji je širu popularnost stekao predstavljanjem AWS Lambda platforme zasnovane na ovom modelu krajem 2014 godine. Model je organizovan je tako da kod na serverskoj strani piše razvojaoc u obliku funkcija koje su bez stanja, pokreću se na osnovu događaja i njihovo izvršavanje u potpunosti kontroliše isporučioc platforme.

Dakle funkcija kao servis platforme omogućavaju izvršavanje koda bez podešavanja servera ili na njima postavljenih serverskih procesa koji su dugog životnog veka. Kod je organizovan u obliku funkcija. Funkcije se grupišu u izvršne jedinice, često kontejnere, o čijem se pokretanju, životnom veku i upravljanju prilikom izvršavanja stara platforma. Ovde se ogleda i razlika u odnosu na model platforma kao servis, kod koga razvijaoc nema kotrolu upravljanja serverom, ali ima nad serversim procesima putem odgovarajućih servisa.

Posmatrano sa strane koda, funkcije na ovom modelu su regularne i nisu zavisne od konkretnog programskog jezika ili razvojnog okvira. Isporučioci platformi u oblaku omogućavaju pisanje funkcija u većini popularnih programskih jezika kao što su Java, C#, Javascript, Python i drugi. Sa druge strane postavljanje u produkciju se dosta razlikuje od ostalih modela, na taj način što se kod aplouduje na posebno određene lokacije na plaformi isporučioca koje zatim platforma koristi za instanciranje izvršnih jedinica. Rezervisanje infrastrukturnih resursa za izvršavanje i upravljanje procesima obavlja platforma na automatizovan način.

Funkcije se pokređu putem događaja koji mogu biti HTTP zahtev, tajmer, ili događaj koji si dogodio na nekim od drugih servisa na platformi u oblaku. Veliki isporučioci omogućavaju veliki broj različitih tipova događaja na osnovu ostalih servisa koje imaju u ponudi. Primeri bi bili upis u neku od baza podataka na platformi, pristigla poruka u redu za obradu i drugi.

Na slici 4 prikazan je uprošćen diagram arhitekture platforme na modelu funkcija kao servis. Nakon pristizanja novog zahteva, zadatak platforme je da zahtev primi u red i zatim ga prosledi odgovarajućoj aktivnoj izvršnoj jedinici ili alocira novu izvršnu jedinicu za njegovu obradu. Platforma mora pokrenuti dovoljan broj instanci izvršne jedinice da opsluži sve pristigle zahteve u zavisnosti od količine saobraćaja. Takođe i dealocira određenu izvršnu jedinicu ukoliko je neaktivna odnosno ukoliko je istekao definisani vremenski period nakon poslednjeg obrađenog zahteva. Na ovaj način horizontalno skaliranje je prebačeno na plaftormu, i odvija se na automatizovan način, bez bilo kakvog podešavanja razvijaoca.



Slika 4 Arhitektura platforme zasnovane na modelu funkcija kao servis

2.4.2 Osobine

Postoji veći broj osobina koje karakterišu model funkcija kao servis. Njihovo poznavanje omogućava razvijaocu poređenje različitih platformi i bolji uvid prilikom izbora i korišćenja odgovarajuće platforme.

* Čuvanje stanja – za funkcije na ovom modelu se kaže da su bez stanja (eng. Stateless) i podaci se čuvaju samo u promenljivama funkcije. Odnosno nema garancija da se stanje može čuvati između više različitih poziva funkcije, i za ove potrebe moraju se koristiti eksterni sistemi za skladištenje. Ipak, neki od isporučioca imaju u ponudi i mehanizme koji čuvaju stanje i olakšavaju ulančane pozive i integraciju više funkcija.
* Performanse – različiti činioci i limiti utiču na performanse koda na ovom modelu, od broja konkurentnih zahteva, do maksimalne veličine memorije i procesorskih resursa za jedan poziv. Pored ovih platforme često imaju i vremensko ograničenje trajanja jednog zahteva, nakon čega se procesiranje zahteva zaustavlja. Takođe, vreme od pristizanja zahteva do početka izvršavanja funcije se može razlikovati u slučajevima kada se koristi postojeća izvršna jedinica, što nazivamo topli start (eng. Warm start) i kada se alocira nova, odnosno hladni start (eng. Cold start). Ova ralika može iznositi od nekoliko milisekundi do čak nekoliko sekundi.
* Naplata – naplata zavisi od količine i vremenske raspoređenosti zahteva i vrši se samo za utrošene resurse prilikom izvršavanja. Sve veće platforme imaju mogućnost skaliranja do nule u periodima kada nema zahteva za obradu i u tim trenucima razvijaoc nema troškova.
* Otvorenost koda – iako je većina platformi zatvorenog koda, postoje i one otvorenog koda kao što su OpenFaaS i Fission. Ove platforme je moguće hostovati i na lokalnim serverima na kojima postoji Kubernetes ili drugi odgovarajući sistemi za orkestraciju kontejnera. Od velikih javnih isporučioca platforme koje su otvorenog koda su IBM OpenWhisk i Microsoft Azure Functions čiji je kod javno dostupan na GitHub servisu pod Apache, odnosno MIT licencom.
* Bezbednost – bezbednost na ovom modelu ima dosta sličnosti sa modelom platforma kao servis. S obzirom da nema servera ili virtualnih mašina, nije potrebno održavanje sigursnosnih zakrpa na tom nivou. Isporučioci platformi u oblaku često imaju posebne servise za upravljanje nalozima i dozvolama na platformi poput AWS IAM ili Azure AD i oni imaju primenu i na ovde. Podešavanju bezbednosti na ovim servisima treba prisupiti pažljivo, bar na produkcionim nalozima.
* Lokalni razvoj, nadgledanje i testiranje – neki od isporučioca imaju alate koji olakšavaju lokalno pokretanje i debagovanje funkcija. Kod drugih debagovanje i nadgledanje je moguće kroz upis u dnevnik datoteke funkcija. Jedinično testiranje funkcija je jednostavno s obzirom da su one samo kod. Sa druge strane, integraciono testiranje složenih aplikacija na ovom modelu može biti veliki izazov.

2.4.3 Prednosti i nedostaci

Neke od prednosti ovog modela proizilaze direktno iz njegovog načina rada. Zbog automatskog skaliranja i ne korišćenja resursa u neaktivnom stanju, razvijaoc može imati veće uštede prilikom naplate servisa. Najbolji primeri su aplikacije kod kojih postoje periodi kada nema zahteva ili je saobraćaj nekonzistentan. Takođe, u odnosu na sisteme zasnovane na virtualnim mašinama i kontejnerima, može se govoriti o uštedi u operacionim aktivnostima, zbog toga što upravljanje infrastrukturom obavlja isporučioc. Prednost ovog modela kod razvoja kompletno novih aplikacija je i to što se razvijaoc može više fokusirati na sam dizajn arhitekture i aplikativni kod, i tako brže doći do prvih upotrebljivih verzija aplikacije.

Sa druge strane, zbog vremenskog ograničenja za izvršavanje zahteva i ne postojanja stanja na serverskoj strani, ovaj model nije adekvatan za neke vrste aplikacija. Slično se može reći i u slučajevima u kojima postoji potreba razvijaoca za specifičnom kontrolom i konfiguracijom infrastrukture. Generalno, aplikacije na ovom modelu su manje prenosive u odnosu na standardne virtualne mašine ili kontejnere. Događaji koji prouzrokuju pokretanje funkcija mogu se bazirati na drugim servisima platforme u oblaku, što može dovesti do veće zavisnosti razvijaoca od konkretnog isporučioca platforme.

3 Platforme na modelu funkcija kao servis

U ovom poglavlju biće dat kratak pregled platformi na modelu funkcija kao servis. U drugom delu biće detaljno predstavljena platforma Azure Functions i njene karakteristike, razvoj funkcija i aplikacija i postavljanje na platformu u oblaku.

3.1 Pregled

Jedna od prvih i najpoznatijih platformi na ovom modelu je svakako Amazon AWS Lambda. Ova platforma omogućava pisanje funkcija bez stanja u jezicima Java, Python, C#, Node.js, Go, Ruby i PowerShell. Funkcije se pokreću na osnovu događaja na drugim servisima na AWS platformi u oblaku, kao što su postavljanje dokumenta na S3 servis za skladištenje, prosleđen HTTP zahtev sa API Gateway servisa i mnogi drugi. Za debagovanje i nadgledanje aplikacija mogu se koristiti AWS X-Ray i Amazon Cloud Watch servisi. Pored toga dostupan je AWS Serverless Application Repository gde je moguće pronaći gotove aplikacije ili funkcije razvijene od strane zajednice koje je moguće koristiti.

Google Cloud Functions je funkcija kao servis platforma kompanije Google. Podržani jezici su Go, Node.js, Java i Python. Platforma ima integrisane funkcionalnosti za debagovanje, nadgledanje i upis u dnevnik datoteke. Funkcije se pokreću na osnovu HTTP zahteva ili događaja na servisima Google Cloud platforme. Kod funkcija je moguće modifikovati lokalno i preko veb portala platforme. Platforme velikih ispručioca na ovom modelu upotpunjuju IBM Cloud Functions i Oracle Cloud Functions.

Najznačajniji projekat otvorenog koda na ovom modelu je Apache OpenWhisk. Platfomu je moguće hostovati na Kubernetes klasteru na lokalnim serverima ili na nekim od Kubernetes servisa na platformama u oblaku. Takođe IBM Cloud Functions je bazirana na ovom projektu i moguće je koristiti kao gotovo rešenje na platformi ovog isporučioca. Podržani su jezici Go, Java, Javascript, C#, Python, PHP, Ruby i Swift. Projekat pruža i alate komandne linije za kreiranje i debagovanje funkcija, kao i za postavljanje i upravljanje funkcijama u produkciji. Na sličan način funkcionišu i druge platforme otvorenog koda, kao što su OpenFaas, Knative i Kubeless.

3.2 Azure Functions

Azure Functions je servis na platformi Microsoft Azure, zasnovan na modelu funkcija kao servis. Zajedno sa Logic Apps i Event Grid servisima čini grupu servisa koji omogućavaju arhitekturu „bez servera“ na ovoj platformi i u narednom delu biće predstavljen njegov detaljniji pregled.

3.2.1 Izvršno okruženje i planovi korišćenja

Azure Functions okruženje zaduženo je za izvršavanje funkcija na plaftormi u oblaku, a alternativno moguće je i njegovo hostovanje na lokalnim serverima. Razvijeno je kao softver otvorenog koda u vlasništvu kompanije Microsoft i prvi put predstavljeno u Januaru 2017. godine. U trenutku pisanja ovog rada aktuelna je bila stabilna verzija 3.1. izvršnog okruženja. Za razvoj funcija inicijalno bili su podržani jezici C#, Javascript i F#, a kasnije verzije donele su podršku za druge jezike. U tabeli 1 prikazana je podrška jezika i njihovih radnih okvira po verzijama okruženja, a postoji mogućnost dodavanja podrške za nove jezike korišćenjem jezičkih proširenja (eng. Language Extensibility).

|  |  |  |  |
| --- | --- | --- | --- |
| Jezik | 1.x | 2.x | 3.x |
| C# | Da (.NET 4.7) | Da (.NET Core 2.2) | Da (.NET Core 3.1) |
| Javascript | Da (Node 6) | Da (Node 10 i 8) | Da (Node 12 i 11) |
| F# | Da (.NET 4.7) | Da (.NET Core 2.2) | Da (.NET Core 3.1) |
| Java | Ne | Da (Java 8) | Da (Java 11 i 8) |
| PowerShell | Ne | Da (PowerShell 6) | Da (PowerShell 7 i 6) |
| Python | Ne | Da (Python 3.7 i 3.6) | Da (Python 3.8, 3.7 i 3.6) |
| Typescript | Ne | Da | Da |

Tabela 1. Podržani jezici [3]

Kao i kod većine ostalih servisa na platformi Azure, ovaj servis se oslanja na planove za korišćenje (eng. Azure App Service plans) koji definišu resurse koje je moguće koristiti, region dostupnosti, detalje naplate i dodatna moguća podešavanja hostovanja. Ponudjena su tri plana za korišćenje Azure Functions servisa korisnicima i oni su redom: osnovni konzumacioni plan (eng. Consumption plan), premium (eng. Premium plan) i namenski plan (eng. Dedicated plan). Prva dva se odnose na arhitekturu “bez servera”, što bi značilo da se instance funkcija dinamički dodaju i uklanjaju u zavisnosti od broja dogadjaja koji prouzrokuju njihovo okidanje u tom trenutku. Naplata se odvija u zavisnosti od ukupnog broja poziva, vremenskog trajanja i količine memorije prilikom izvršavanja. Treba napomenuti i da različiti planovi korišćenja preciziraju i maksimalno vreme izvršavanja (eng. Execution timeout), i da kompanija Microsoft, slično kao i ostali isporučioci ovih servisa, tvrdi da je omogućeno neograničeno skaliranje. Za razliku od njih namenski plan omogućava da se izvršno okruženje doda na postojeće virtualne mašine korisnika na Azure platformi, a naplata se u tom slučaju odvija kao kod ostalih servisa, bazirano na vremenu rezervisanosti.

3.2.2 Funkcije i aplikacije funkcija

Validne funkcije se sastoje od dve datoteke, prve koja sadrži kod za izvršavanje u jeziku koji je korisnik odabrao i druge pod nazivom function.json koja sadrži konfiguracioni kod u formatu JSON. U ovoj datoteci definisan je okidač, sva vezivanja i dodatni konfiguracioni parametri okruženja. Za kompilirane programske jezike moguće je automatski generisati function.json datoteku prilikom faze kompilacije, dok se za interpretirane jezike ona mora posebno napisati.

Funkcije su najčešće grupisane u aplikacije funkcija (eng. Function App), iako se mogu razvijati i koristiti i samostalno. Od verzije 2.0 izvršnog okruženja funkcije koje su deo iste aplikacije moraju biti napisane u istom programskom jeziku i koristiti istu verziju okruženja. Dok sa druge strane aplikacije omogućavaju lakše upravljanje grupom funkcija i njihovo postavljanje i podešavanje na Azure plaftormi. Aplikacije imaju definisanu strukturu direktorijuma kako bi postavljanje na Azure platformu i izvršavanje bilo uniformno i ona mora biti poštovana bez obzira na programski jezik i radni okvir razvoja.



Slika 2 Struktura aplikacije funkcija

Na slici 1 data je organizacija tipične aplikacije funkcija. U čvornom direktorijumu se nalazi host.json datoteka sa konfiguracionim podešavanjima aplikacije, svaka od funkcija je u posebnom poddirektorijumu, kao i opcioni deljeni kod, dok se u bin direktorijumu nalaze izvršne datoteke. U zavisnosti od programskog jezika mogu biti definisana dodatna pravila u strukturi, više informacije o pojedinačnim jezicima se može naći na veb lokaciji [4].

3.2.3 Lokalni razvoj

Za potrebe pokretanja funkcija na lokalnom računaru potrebno je instalirati alat Azure Functions Core Tools [5] kao i radni okvir za programski jezik koji je izabran, njihove trenutne verzije su prikazane u tabeli 1. Alternativno za pokretanje funkcija mogu se koristiti dodataci za razvojno okruženje Visual Studio i Visual Studio Code, a za pisanje koda samih funkcija i druga okruženja koja olakšavaju rad u izabranom jeziku.

U nastavku biće dat primer u jeziku C# i korišćenjem Azure Functions Core Tools [5] alata iz komandne linije i generičkog tekstualnog editora. Na slici 2 data je komanda kojom kreira se C# biblioteka klasa (eng. Class library) koja predstavlja aplikaciju funkcija.



Slika 3 Kreiranje aplikacije funkcija

Funkcije dodajemo kreirajući datoteke sa ekstenzijom „.cs“ za kod funkcija u odgovarajućim direktorijumima. Na slici 3 dat je primer koda funkcije sa okidačem na HTTP zahtev. Funkcija čita sadržaj parametra ime i na osnovu njega vraća odgovarajuću poruku u telu HTTP odgovora. Iz primera se može videti da se okidači mogu specifikovati kao C# atributi parametara funkcije. Više reči o okidačima i vezivanjima biće u narednoj sekciji.



Slika 4 Primer funkcije sa Http okidačem

Kompiliranje i pokretanje aplikacije se vrši komandom na slici 4. Nakon toga funkcija je dostupna lokalno za pozivanje na predefinisanom portu 7071, odnosno na lokaciji <http://localhost:7071/api/httpPrimer>.



Slika 5 Kompilacija i pokretanje aplikacije

Na ovaj način automatski je izgenerisana i konfiguraciona datoteka function.json i njen sadržaj dat je na slici 5.



Slika 6 Primer function.json datoteke

3.2.4 Okidači i vezivanja

Vezivanja (eng. Bindings) definišu načine na koji funkcija komunicira sa spoljašnjim svetom ili ostalim servisima razvijaoca na Azure platformi. Funkcija može imati veći broj vezivanja i ona mogu biti ulazna, izlazna ili dvosmerna. Podaci iz ulaznih vezivanja su prilikom izvršavanja dostupni kao parametri funkcije, dok se na izlazna vezivanja mogu slati podaci u telu funkcije ili kao njena povratna vrednost. Primeri ulaznog vezivanja bi bili tajmer, HTTP zahtev, upis koji se dogodio na Blob storage servisu, ulazna poruka u Queue storage servisu, dogadjaj na Cosmos DB bazi ili na drugim servisima. Izlazno vezivanje može biti prosledjivanje rezultata funkcije na ove ili druge servise. U tabeli 2 data su svi podržani tipovi vezivanja.

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Servis | 1.x | 2.x i više | Okidač | Ulazno vezivanje | Izlazno vezivanje |
| Blob storage | ✔ | ✔ | ✔ | ✔ | ✔ |
| Cosmos DB | ✔ | ✔ | ✔ | ✔ | ✔ |
| Dapr |  | ✔ | ✔ | ✔ | ✔ |
| Event grid | ✔ | ✔ | ✔ |  | ✔ |
| Event hubs | ✔ | ✔ | ✔ |  | ✔ |
| HTTP | ✔ | ✔ | ✔ |  | ✔ |
| IoT hubs | ✔ | ✔ | ✔ |  | ✔ |
| Kafka |  | ✔ | ✔ |  | ✔ |
| Mobile Apps | ✔ |  |  | ✔ | ✔ |
| Notification Hubs | ✔ |  |  |  | ✔ |
| Queue storage | ✔ | ✔ | ✔ |  | ✔ |
| RabbitMQ |  | ✔ | ✔ |  | ✔ |
| SendGrid | ✔ | ✔ |  |  | ✔ |
| Service Bus | ✔ | ✔ | ✔ |  | ✔ |
| SignalR |  | ✔ |  | ✔ | ✔ |
| Table storage | ✔ | ✔ |  | ✔ | ✔ |
| Tajmer | ✔ | ✔ | ✔ |  |  |
| Twillio | ✔ | ✔ |  |  | ✔ |

Tabela 2 Podržani okidači i vezivanja

U function.json datoteci vezivanja su definisana u posebnom nizu pod nazivom vezivanja (eng. Bindings). Svaki element niza minimalno sadrži parametre tip (eng. Type) servisa za koji se definiše vezivanje, naziv (eng. Name), smer (eng. Direction) i tip podataka (eng. DataType) koje vezivanje očekuje. Ukoliko se koristi jezik C# moguće je specifikovanje vezivanja preko strogo tipiziranih C# atributa u kodu funkcije, da bi se na osnovu njih u fazi kompilacije generisala odgovarajuća sekcija u datoteci function.json.

Poseban tip ulaznog vezivanja koja prouzrokuje izvršavanje funkcije je okidač i svaka funkcija mora imati tačno jedan okidač. Okidači u svom nazivu moraju imati nastavak „Trigger“ kako bi se razlikovali od ostalih ulaznih vezivanja. Na slici 6 dat je primer na kome je okidač poruka sa Queue storage servisa. Poruka se zatim formatira i prosledjuje na izlazno vezivanje što je u ovom slučaju nova datoteka na Blob storage servisu. Na slici 7 je data odgovarajuća function.json datoteka za ovu funkciju.



Slika 7 Primer funkcije sa ulaznim i izlaznim vezivanjem



Slika 8 Primer konfiguracije vezivanja

3.2.5 Trajne funkcije

Trajne funkcije (eng. Durable functions) je nadogradnja na platformu Azure Functions koja je otvorenog koda i omogućava pisanje funkcija koje imaju mogućnost čuvanja stanja.

...

3.2.6 Postavljanje na Azure platformu

Za postavljanje lokalno razvijene aplikacije funkcija na platformu potrebno je da prethodno budu kreirani zavisni resursi na platformi i to resursna grupa (eng. Resource group) i nalog za skladištenje (eng. Storage account), a potom i sama aplikacija funkcija. Kreiranje resursa je moguće uraditi na više načina, preko Azure veb portala, korišćenjem Azure alata komandne linije (eng. Azire CLI) ili korišćenjem ARM šablona (eng. ARM template). Na slici 8 prikazano je kreiranje nove aplikacije funkcija iz komandne linije.

Slika 9 Kreiranje aplikacije funkcija na Azure platformi

Prilikom kreiranja moguće je specifikovati region, razvojnu platformu, verziju izvršnog okruženja, naziv aplikacije, naziv resursne grupe, nalog za skladištenje, operativni sistem, plan korišćenja, i da li će se prilikom postavljanja koristiti kod ili priloženi Docker kontejner.

Na nalogu za skladištenje čuvaju se datoteke aplikacije u tri direktorijuma:

* data – u ovom direktorijumu se čuvaju host.json i druge datoteke za konfiguraciju izvršnog okruženja
* LogFiles – čuva log datoteke koje nastaju prilikom izvršavanja
* site – u njemu se nalazi aplikacija funkcija po definisanoj strukturi ili Doker kontejner ukoliko je tako odabrano prilikom kreiranja resursa

Izvršno okruženje koristi ove datoteke za pokretanje novih instanci izvršnih jedinica aplikacije u slučajevima kada je to potrebno. Postavljanje nove verzije aplikacije omogućeno je na više načina od kojih su najkorišćenija dva:

* Postavljanje iz zip datoteke (eng. Zip deployment) – koristi se zip datoteka koja sadrži datoteke aplikacije nakon kompilacije, postavljanje se vrši preko alata komandne linije, Azure portala ili REST API poziva.
* Pokretanje iz paketa (eng. Run from package) – postavljanjem parametra pod nazivom WEBSITE\_RUN\_FROM\_PACKAGE u host.json datoteci, čija se vrednost postavi na link sa paketom za pokretanje koji je javno dostupan na internetu.

Na slici 9 je dat primer komande za postavljanje iz lokalne zip datoteke.



Slika 10 Postavljanje aplikacije na platfomru iz zip datoteke

Osim ovih omogućeno je postavljanje koristeći alate za kontinualnu integraciju Azure DevOps, GitHub Actions, Jenkins i drugih.

4 Razvoj REST servisa Recepti API

4.1 Funkcionalni opis i arhitektura

4.2 Implementacija servisa

4.2.1 Struktura projekta

4.2.2 Baza podataka

4.2.3 API resursi

4.2.4 Implementacija funkcija

5 Gradjenje, hostovanje i testiranje servisa Recepti API

5.1 Resursi na platformi u oblaku i postavljanje servisa

5.2 Testiranje

5.3 Nadgledanje i održavanje

6 Zaključak

Reference

|  |  |
| --- | --- |
| [1] | "Gartner," 2019. [Online]. Available: https://www.gartner.com/en/newsroom/press-releases/2019-04-02-gartner-forecasts-worldwide-public-cloud-revenue-to-g. |
| [2] | M. Fowler and J. Lewis, "Microservices," 2014. [Online]. Available: https://martinfowler.com/articles/microservices.html. |
| [3] | "Azure Functions supported languages," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages. |
| [4] | "Azure Functions folder structure," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/functions-reference-node#folder-structure. |
| [5] | "Azure Functions Core Tools," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local. |