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   
Beograd, 2020

*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 4](#_Toc50895755)

[2 Nastanak arhitekture “bez servera” 6](#_Toc50895756)

[2.1 Monolitne arhitekture, SOA 6](#_Toc50895757)

[2.2 Mikroservisna arhitektura, kontejneri 6](#_Toc50895758)

[2.3 Računarstvo u oblaku, tipovi i modeli servisa 6](#_Toc50895759)

[2.4 Arhitektura “bez servera” 6](#_Toc50895760)

[2.4.1 Model izvršavanja funkcija kao servis 6](#_Toc50895761)

[3 Servisi zasnovani na modelu funkcija kao servis 6](#_Toc50895762)

[3.1 Azure Functions 6](#_Toc50895763)

[3.1.1 Izvršno okruženje i planovi korišćenja 7](#_Toc50895764)

[3.1.2 Funkcije i aplikacije funkcija 8](#_Toc50895765)

[3.1.3 Lokalni razvoj 9](#_Toc50895766)

[3.1.4 Okidači i vezivanja 11](#_Toc50895767)

[3.1.5 Postavljanje na Azure platformu 13](#_Toc50895768)

[3.2 Ostali servisi 16](#_Toc50895769)

[4 Razvoj REST servisa Recepti API 16](#_Toc50895770)

[4.1 Funkcionalni opis i arhitektura 16](#_Toc50895771)

[4.2 Implementacija servisa 16](#_Toc50895772)

[4.2.1 Struktura projekta 16](#_Toc50895773)

[4.2.2 Baza podataka 16](#_Toc50895774)

[4.2.3 API resursi 16](#_Toc50895775)

[4.2.4 Implementacija funkcija 16](#_Toc50895776)

[5 Gradjenje, hostovanje i testiranje servisa Recepti API 16](#_Toc50895777)

[5.1 Resursi na platformi u oblaku 16](#_Toc50895778)

[5.2 Kontinualna integracija 16](#_Toc50895779)

[5.3 Nadgledanje i održavanje 16](#_Toc50895780)

[5.4 Testiranje 17](#_Toc50895781)

[6 Zaključak 17](#_Toc50895782)

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 potpuno 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”

Aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa

2.1 Monolitne i slojevite arhitekture

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 može uticati na dostupnost cele aplikacije. Horizontalno skaliranje moguće je jedino postavljanjem više identičnih instanci cele aplikacije ispred rasporedjivača opterećenja (eng. Load balancer).

2.2 Mikroservisna arhitektura

Servisno orijentisana arhitektura (SOA) nastala je sa ciljem da eliminiše neke od nedostataka monolitnih aplikacija, a pre svega da poveća njihovu proširivost.

2.3 Računarstvo u oblaku, tipovi i modeli servisa

2.4 Računarstvo “bez servera”

2.4.1 Model izvršavanja funkcija kao servis

3 Platforme zasnovane na modelu funkcija kao servis

3.1 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.

3.1.1 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* [2]

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.

3.1.2 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 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 1 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 [3].

3.1.3 Lokalni razvoj

Za potrebe pokretanja funkcija na lokalnom računaru potrebno je instalirati alat Azure Functions Core Tools [4] 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 [4] alata iz komandne linije i običnog tekstualnog editora. Na slici 2 data je komanda kojom kreira se C# biblioteka klasa (eng. Class library) koja predstavlja aplikaciju funkcija.



Slika 2 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 3 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 4 Kompilacija i pokretanje aplikacije

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



Slika 5 Primer function.json datoteke

3.1.4 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 6 Primer funkcije sa ulaznim i izlaznim vezivanjem



Slika 7 Primer konfiguracije vezivanja

3.1.5 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 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 8 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 Doker 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 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 9 Postavljanje aplikacije na platfomru iz zip datoteke

Osim ovih omogućeno je postavljanje koristeći alate za kontinualnu integraciju Azure DevOps, GitHub Actions ili dodatka za automatizacioni server Jenkins.

3.2 Ostali servisi

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

5.2 Kontinualna integracija

5.3 Nadgledanje i održavanje

5.4 Testiranje

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] | "Azure Functions supported languages," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/supported-languages. |
| [3] | "Azure Functions folder structure," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/functions-reference-node#folder-structure. |
| [4] | "Azure Functions Core Tools," 2020. [Online]. Available: https://docs.microsoft.com/en-us/azure/azure-functions/functions-run-local. |