Univerza v Ljubljani
Fakulteta za računalništvo in informatiko

Neža Vehovar

Rešitev dogodkovnega prehoda pri večstopenjskih aplikacijah v oblaku

diplomsko delo

visokošolski študijski program
prve stopnje
računalništvo in informatika

Mentor: prof. dr. Vlado Stankovski
Somentor: dr. Matija Cankar

Ljubljana, 2020

Besedilo je oblikovano z urejevalnikom besedil LaTeX.

Fakulteta za računalništvo in informatiko izdaja naslednjo nalogo:

Tematika naloge:

Aplikacije računalništva v oblaku so lahko kompleksne, sestavljene iz večjega števila mikrostoritev in porazdeljene čez večje število ponudnikov. Pri tem gre po navadi za asinhrone načine komuniciranja med posameznimi mikrostoritvami in vključuje tudi proženje različnih dogodkov. Naloga je raziskati principe in arhitekture dogodkovnih prehodov ter ustvariti primer realizacije v kontekstu računalništva v megli.

Za mentorstvo in nasvete se zahvaljujem mentorju izr. prof. dr. Vladu Stankovskemu. S strokovnim znanjem je močno pripomogel h kakovosti diplomskega dela. Zahvaljujem se tudi somentorju dr. Matiji Cankarju za pogovore in nasvete glede teme diplomskega dela in za znanje, potrebno za izvedbo rešitve, o kateri govori delo. Na koncu bi se rada zahvalila še družini za podporo v času študija, še posebej očetu, ki je pokazal posebno zanimanje za temo ter priskrbel edinstveno perspektivo uporabnika na problem, ki ga rešuje delo.

Kazalo

Seznam uporabljenih kratic

kratica	angleško	slovensko
API	Application programming interface	Vmesnik za programiranje aplikacij
AWS	Amazon Web Services	Spletne storitve Amazon
S3	Amazon Simple Storage Service	Amazon storitev za preprosto shrambo
JSON	JavaScript Object Notation	JavaScript objektna oznaka
HTTP	HyperText Transfer Protocol	Protokol za prenos hiperteksta
SQS	Simple Queue Service	Storitev za preprosto vrsto
SNS	Simple Notification Service	Storitev za preprosta obvestila
DevOps	Practice that combines software development (Dev) and IT operations (Ops)	Praksa, ki združuje razvoj programske opreme in IT operacije
YAML	A recursive acronym "YAML Ain't Markup Language"	Rekurzivna kratica za "YAML ni označevalni jezik"
TOSCA	Topology and Orchestration	Specifikacija topologije in or-
OASIS	Specification for Cloud Applications provided by the Organization for the Advancement of Structured Information Standards	kestracije za aplikacije v oblaku, ki jo ponuja organizacija za napredek strukturiranih informacijskih standardov

PIP	Package Manager for Python packages	Upravljalec paketov, napisanih v programskem jeziku Python
SOA	Service-oriented architecture	Storitveno usmerjena arhitektura
FaaS	Function as a service	Funkcija kot storitev
ARN	Amazon resource name	Ime vira pri ponudniku Amazon
QoS	Quality of service	Kakovost storitve

Slovar

Angleško	**Slovensko**
Microservice	Mikro storitev
Vendor-lockin	Odvisnost od ponudnika
Object storage	Objektna shramba
Event handlers	Upravljavci dogodkov
Distributed system	Porazdeljeni sistem
Bucket	Vedro
Container	Zabojnik
Bucket storage	Objektna shramba
Function	Funkcija
Event sources	Viri dogodka
Topic	Tema
Subscription	Naročnina
Policy	Politika delovanja
Function as a service	Funkcija kot storitev
Event grid	Obvestilna mreža
Amazon resource name	Ime amazon vira
Simple notification service	Preprosta storitev za obvestila
Simple storage service	Preprosta storitev za shrambo
Application programming interface	Aplikacijski programski vmesnik
Loosly coupled	Ohlapno povezane
Ansible playbooks	Ansible skript za opravila
Storage account	Račun shrambe

Povzetek

Naslov: Rešitev dogodkovnega prehoda pri večstopenjskih aplikacijah v oblaku

Avtor: Neža Vehovar

Oblačni ponudniki za razvoj aplikacij v oblaku ponujajo neodvisne strukturirane storitve, ki jih imenujemo mikro storitve. Mikro storitve lahko med seboj komunicirajo s pomočjo dogodkov in se povezujejo v aplikacije, ki jim rečemo večstopenjske aplikacije v oblaku. Vsak oblačni ponudnik ima drugače definirane mikro storitve in komunikacijo med njimi, saj standardi za definiranje komunikacije in strukture mikro storitev ne obstajajo, kar pomeni, da lahko mikro storitve med seboj komunicirajo le znotraj ponudnika. Zaradi tega je nastala tako imenovana odvisnost od ponudnika (*ang.* vendor lock-in), ki v določenih primerih uporabnike omejuje pri razvoju kakovostnih aplikacij v oblaku, saj ponudniki oblaka ponujajo različne mikro storitve v različnih kakovostih. Če bi želeli večstopenjsko aplikacijo sestaviti iz mikro storitev različnih ponudnikov, bi za komunikacijo med storitvami potrebovali vmesnik, ki mu v nalogi rečemo dogodkovni prehod. Dogodkovni prehod je tehnologija, uporabljena za prejemanje in preusmerjanje dogodkov ter omogočanje združljivosti med vozlišči. V primeru večstopenjskih aplikacij to pomeni omogočanje komunikacije med mikro storitvami različnih ponudnikov. Diplomska naloga predstavi rešitev, ki omogoča posplošeno komunikacijo med večstopenjskimi storitvami, kar uporabniku olajša sestavo večstopenjskih aplikacij, ki uporabljajo več oblačnih ponudnikov. Rešitev

večstopenjskim aplikacijam omogoča večjo fleksibilnost zaradi razširjene izbire kompatibilnih storitev. Namen diplomskega dela je začeti pogovor o reševanju problema odvisnosti od ponudnika pri večstopenjskih aplikacijah v oblaku ter o težavni komunikaciji med večstopenjskimi storitvami. Področje je še precej novo in mu manjka nekaj gradnikov, kot je dogodkovni prehod za večstopenjske aplikacije, ki bi lahko močno pripomogel h kakovosti aplikacij. Želja je, da bi osnova rešitve spodbudila ponudnike in uporabnike oblaka k razvoju celotne rešitve, ta pa bi pripomogla h kakovosti aplikacij, razvitih pri vseh ponudnikih storitev v oblaku.

Ključne besede: oblak, večstopenjske aplikacije, odvisnost od ponudnika, dogodek.

Abstract

Title: Event gateway solution for multi tier cloud applications

Author: Neža Vehovar

For the development of cloud applications cloud providers offer independently structured services called microservices. Microservices can communicate with each other through events and connect into applications called multi-tier cloud applications. Every cloud provider has differently defined microservices and communication between them, as standards for defining the communication and structure of microservices do not exist. That means microservices can only communicate with each other within the provider. This has led to a so-called vendor lock-in, which in some cases restricts users from developing quality cloud applications, as cloud providers offer different microservices in different qualities. If we wanted to develop a multi-tier application from microservices of different providers, we would need an interface for communication between services, which we in this work call an event gateway. Event gateway is a technology used to receive and redirect events and enable compatibility between nodes. In the case of multi-tier applications, this means enabling communication between the micro-services of different providers. The thesis presents a solution that enables generalized communication between multi-level services, which makes it easier for the user to develop multi-tier applications that use multiple cloud providers. The solution allows multi-tiered applications more flexibility, thanks to an expanded selection of compatible services. The purpose of the thesis is to

start a conversation about solving the problem of vendor lock-in in multi-tier applications in the cloud, and about the difficult communication between multi-tier services. The field is still quite new and lacks some building blocks, such as an event gateway for multi-tier applications that could greatly contribute to the quality of applications. The desire is that the base solution would encourage cloud providers and users to develop the entire solution, which would contribute to the quality of applications developed on all cloud platforms.

Keywords: cloud, multi tier applications, vendor lock-in, event.

Poglavje 1

Uvod

Dogodki se v računalništvu uporabljajo že od samega začetka in so namenjeni zaznavanju zunanje dejavnosti, ki jo program prepozna in obravnava [6]. O dogodku v računalništvu govorimo, ko programska oprema prepozna akcijo iz zunanjega okolja. Akcija je lahko premik miške, pritisk tipke, sporočilo iz senzorja ali kakršen koli dogodek, na katerega se program lahko odzove [7]. Programom, ki rokujejo z dogodki, pravimo upravljavci dogodkov (*ang.* event handlers). Takšni programi po navadi tečejo v neskončni zanki, v kateri preverjajo, ali je bil zaznan nov dogodek. Ko je dogodek zaznan, se mora programska oprema nanj ustrezno odzvati. Pri kompleksnih aplikacijah lahko v tem koraku uporabimo dogodkovni prehod, ki je del programske opreme in je namenjen prejemanju in preusmerjanju dogodkov ter omogočanju združljivosti med vozlišči. To nam omogoča pošiljanje dogodka na želeno končno točko, ki lahko izvede ustrezen odziv na dogodek.

Pri storitveno usmerjeni arhitekturi aplikacij ali SOA so dogodki zasnovani za sporočanje znotraj aplikacije in so tesno povezani z arhitekturo aplikacije in zato točno določeni že od stopnje snovanja. Storitveno usmerjene aplikacije so zato monolitne in so v začetku lahko razvojno preproste, vendar se s širjenjem močno zapletejo in ustvarijo veliko pritiska in stroškov za podjetje. Aplikacije so težke za razumevanje in pogosto zahtevajo sodelovanje celotne razvojne ekipe zaradi testne povezanosti komponent. Oblak

pa uporablja drugačno arhitekturo, in sicer mikro storitve, ki so med seboj neodvisne in se v aplikacijo povezujejo z natančno določenimi aplikacijskimi programskimi vmesniki (*ang.* API). Pri oblačnih mikro storitvah so dogodki in upravljavci dogodkov element, ki povezuje drugače neodvisne storitve, torej služijo za medsebojno komunikacijo med mikro storitvami in so znotraj ponudnika strogo definirani. Arhitektura mikro storitev nam omogoča fleksibilno in modularno arhitekturo aplikacij, vendar pomeni kompleksno snovanje dogodkov zaradi velikega števila mikro storitev. Kljub zahtevnosti povezave med mikro storitvami njihova priljubljenost narašča, saj je oblačni ponudnik tisti, ki prevzame odgovornost za zahtevno arhitekturo povezav in uporabniku omogoči lažjo razširljivost aplikacije.

Aplikacijam, ki so sestavljene iz več mikro storitev, pravimo večstopenjske aplikacije. Razvoj večstopenjske aplikacije je v začetku bolj zahteven, saj pomeni snovanje porazdeljene aplikacije ali sistema, vendar je kasneje bolj razširljiv [12]. Komunikacija med mikro storitvami je v večstopenjski aplikaciji dobro definirana, dokler uporablja storitve znotraj ponudnika. Na težave pa naletimo, ko želimo bolj fleksibilno prepošiljanje dogodkov na mikro storitve drugih ponudnikov, zasebni oblak ali druge neodvisne storitve (na primer elektronska pošta). To pomeni, da je uporabnik pri snovanju večstopenjskih aplikacij odvisen od ponudnika, kar vpliva na kakovost končnega izdelka. Vsak ponudnik ima svoje prednosti in slabosti, zato morajo uporabniki dobro oceniti, kateri ponudnik jim bo ustrezal, saj je kasnejša migracija zelo drag in kompliciran proces.

Uporabnik pa lahko odvisnost od ponudnika pri večstopenjskih aplikacijah rešuje tudi tako, da sam definira dogodkovni prehod med mikro storitvami. Razvoj takšnega dogodkovnega prehoda, ki je tudi zanesljiv in razširljiv, je zahteven in bi posameznemu podjetju povzročil velike stroške. Težava je v tem, da ima vsaka mikro storitev drugačne točke, na katere lahko naslovi dogodek, katere pa so pogosto zelo omejene. Zato je potrebna raziskava, katera od tehnologij je najbolj primerna za prenos dogodka. Prav tako je treba upoštevati varnostne mehanizme, ki so potrebni za varno uporabo

mikro storitev, razširljivost razvite rešitve za večje število prenosov dogodkov ter kakovost storitve (*ang.* QoS).

Zato bi bil dogodkovni prehod, ki omogoča komunikacijo med sicer težko povezljivimi mikro storitvami v oblaku, zelo zaželeno orodje za marsikatero podjetje. Če bi bil ponujen kot storitev, pa bi to pomenilo, da podjetja ne rabijo razviti svojega dogodkovnega prehoda, temveč ga lahko, tako kot druge oblačne storitve, plačujejo po porabi.

Poglavje 2

Problem in metode reševanja

2.1 Problem

Problem, ki ga rešuje diplomsko delo, je težavno postavljanje aplikacij čez več oblakov, h kateremu močno pripomore problem odvisnosti od ponudnika. Mikro storitve v aplikacijah med seboj komunicirajo s pomočjo dogodkov, katere si lahko predstavljamo tudi kot podatke, ki se med mikro storitvami pošiljajo preko mreže. Vsaka mikro storitev ima na voljo določene končne točke, kamor lahko naslovi dogodke. Znotraj ponudnika so naslovniki dogodkov dobro zasnovani, ko pa pride do pošiljanja dogodkov med ponudniki, pa nastopijo težave. Dogodke si želimo pošiljati med ponudniki, ponudnikov pa je veliko in vsak ima svojo močne in šibke točke. Zato si v določenih primerih želimo združiti močne točke več ponudnikov, da dosežemo optimalno rešitev aplikacije v oblaku. V tem primeru potrebujemo za komunikacijo mikro storitev med ponudniki dogodkovni prehod, ki omogoča kompatibilnost med začetno in končno točko komunikacije. Dogodkovni prehod tako omogoča hibridne rešitve med ponudniki, ki sicer niso zasnovani kot hibridni oblačni ponudniki.

Cilj diplomskega dela je zasnovati sistem, ki je namenjen prenašanju dogodkov med težko povezljivimi mikro storitvami v oblaku. Končna rešitev bo implementirana kot odjemalni program, ki bo uporabniku omogočal na-

stavitev dogodkovnega prehoda za lastne potrebe. Odjemalni program bo uporabnika vodil čez nastavitev dostopnih podatkov, nastavitev gostitelja dogodkovnega prehoda in nastavitev podatkov o začetni in končni točki prehoda, ter od uporabnika ne bo zahteval nobenega predhodnega znanja. Tako bo rešitev uporabniku močno skrajšala čas, ki je potreben za postavitev aplikacije čez več oblakov. Cilj dela je tudi spodbuditi razvojnike, da pri snovanju aplikacije upoštevajo vse mikro storitve in ne le mikro storitve enega ponudnika. To lahko v prihodnosti pomeni bolj kakovostne aplikacije, kar je motivacija za izdelavo tega diplomskega dela. Dogodkovni prehod bo od uporabnika zahteval le, da zna poiskati zahtevane podatke o vpletenih mikro storitvah na platformi ponudnika. Od tam naprej pa bo odjemalni program sam postavil vse potrebne gradnike za delujočo aplikacijo s pomočjo programskih knjižnic za upravljanje s storitvami oblačnih ponudnikov.

2.2 Pregled dosedanjih metod in poskusov

Da lahko bolje razumemo področje, moramo poiskati in raziskati rešitve, ki trenutno obstajajo za reševanje danega problema. Tako lahko ugotovimo, kaj so pomanjkljivosti dosedanjih rešitev in metod in to upoštevamo pri snovanju nadaljnjih rešitev.

2.2.1 Analiza serverless framework tehnologije

Ogrodje serverless je tehnologija, ki je najbolj podobna rešitvi, opisani v diplomskem delu, in je namenjena pomoči uporabnikom pri rokovanju z aplikacijami v oblaku. Zagotavlja enostavno razvijanje aplikacije, ki je lahko postavljena na oblak katerega koli ponudnika, lokalno testiranje ipd. in je na splošno v veliko pomoč pri razvijanju vseh vrst aplikacij v oblaku. Na spletni strani orodja oglašujejo, da orodje omogoča delo z dogodki v oblaku, vendar te funkcionalnosti še niso na voljo uporabnikom. Na razvojni strani omenijo, da je orodje še v fazi obsežnega razvoja, medtem ko je bila zadnja sprememba narejena pred dvema letoma. Iz tega lahko sklepamo, da orodje

v kratkem še ne bo funkcionalno.

2.2.2 Microsoft Azure event grid tehnologija

Event grid tehnologija je del oblačnega ponudnika Microsoft Azure in je namenjena preusmerjanju dogodkov med Microsoft Azure mikro storitvami. Z implementacijo podobnih storitev pri vseh ponudnikih bi se lahko znebili potrebe po dogodkovnem prehodu, saj bi to pomenilo, da ponudniki oblaka omogočajo povezljivost z zunanjimi storitvami. Vse mikro storitve znotraj Microsoft Azure, ki generirajo dogodke, lahko dogodke preusmerijo na mikro storitev event grid. Od tam naprej pa lahko event grid pošlje dogodek na vrsto različnih končnih točk. Uvedba take storitve je pametna odločitev, saj se izognemo definiranju velikega števila končnih točk za vsako mikro storitev. Namesto tega ima veliko število končnih točk le event grid, medtem ko lahko ostale mikro storitve pošiljajo dogodke le na event grid. Možnosti končnih točk event grid storitve vsebujejo tudi možnost preusmerjanja dogodkov s pomočjo tehnologije webhook, kar pomeni, da lahko dogodek naslovimo na funkcijo, ki se nahaja na drugem uporabniškem računu ali pri drugem ponudniku. To nam močno olajša pošiljanje dogodkov na končne točke, ki se ne nahajajo znotraj ponudnika. Tehnologija event grid bo uporabljena kot del rešitve dogodkovnega prehoda.

2.2.3 Specifikacija CloudEvents

CloudEvents je specifikacija za dogodke v oblaku. Za lažji razvoj rešitev v oblaku so standardi zelo koristni, to lahko v nadaljevanju opazimo pri delu z dogodki različnih ponudnikov, ki še ne delujejo preko standarda. Cilj tehnologije je zasnovati format dogodkov, ki bi bil posplošen za vse in čez vse ponudnike in tako omogočal boljšo povezljivost. CloudEvents v svoji dokumentaciji poudarja težavo, ki nastane pri snovanju dogodkov, saj jih vsak ponudnik definira na svoj način. Poenoteni dogodki bi optimizirali razvoj in omogočili prenosljivost programske opreme. CloudEvents je prav

tako še v procesu razvoja [4]. Če bi bil CloudEvents standard že sprejet in uporabljen iz strani ponudnikov Amazon in Microsoft, bi nam to močno olajšalo implementacijo dogodkovnega prehoda.

2.3 Pomanjkljivosti dosedanjih poskusov

Pri omenjenih rešitvah opazimo, da je to področje na splošno še precej nedotaknjeno, rešitve, ki pa že obstajajo, pa so specifične za ponudnika oblaka ali so še v razvoju. Kljub temu, da ob internetnem iskanju dobimo ogromno rezultatov, objav in člankov na temo odvisnosti od ponudnika, ki opozarjajo na težavo, še vedno ni velikega napredka pri reševanju problema.

Uporabniki predvsem poročajo o težavni migraciji na oblak drugega ponudnika, za kar sicer obstajajo orodja, vendar so daleč od trivialnih. Posplošeni standardi bi močno olajšali delo, ki ga mora opraviti orodje za migracijo, prav tako pa bi uporabniku prihranili veliko časa, saj ponovno učenje tehnologij drugega ponudnika ne bi bilo potrebno [14]. Migracija pa v nekaterih primerih ne bi bila potrebna, če bi obstajal dober vmesnik za komunikacijo mikro storitev, ki bi omogočal večstopenjske aplikacije čez več oblakov. Posplošeni standardi bi olajšali tudi delo, ki ga mora opraviti vmesnik za komunikacijo mikro storitev, na primer enotno definirano strukturo dogodko. Priložnosti za izboljšave je veliko, v tem delu pa se osredotočimo na vmesnik za komunikacijo med mikro storitvami.

Poglavje 3

Pregled tematike

Poglavje pregled tematike pokriva področja in tehnologije, ki so omenjena v nadaljevanju dela in so potrebne za implementacijo in razumevanje vmesnika za komunikacijo med mikro storitvami oziroma dogodkovnega prehoda.

3.1 Dogodkovni prehodi

Dogodkovni prehod je del programske opreme, ki izvaja neskončno zanko, v kateri preverja, ali je prejel nove dogodke. Na dogodke se nato na predpisan način odzove. To je lahko prepošiljanje dogodka na drugo končno točko, časovno načrtovano prepošiljanje dogodka, filtriranje dogodkov, analiza dogodkov in podobno [11]. Dogodkovni prehodi so koristni tudi za združevanje nekompatibilnih storitev ter omogočanje komunikacije med njimi, saj imajo definirano vrsto vhodov in izhodov, ki med seboj niso odvisni. Dogodek si lahko predstavljamo kot avtomobil, ki se po cesti pripelje do križišča (tj. dogodkovnega prehoda) in nato lahko zavije na katero koli od n cest.

Arhitektura aplikacij, ki upravljajo z dogodki, postaja vse bolj razširjena ravno zaradi oblačnih storitev. Sami ponudniki storitev ponujajo različna orodja za zaznavanje, analizo in spremljanje dogodkov, ne omogočajo pa robustnega preusmerjanja dogodkov na različne končne točke. Torej ne omogočajo funkcionalnosti dogodkovnega prehoda, ki jo mnogi potrebujejo.

Podjetja si zato pogosto sama razvijejo dogodkovne prehode za lastne potrebe.

Glavne zahteve dogodkovnega prehoda, ki jih želimo izpolniti, so:

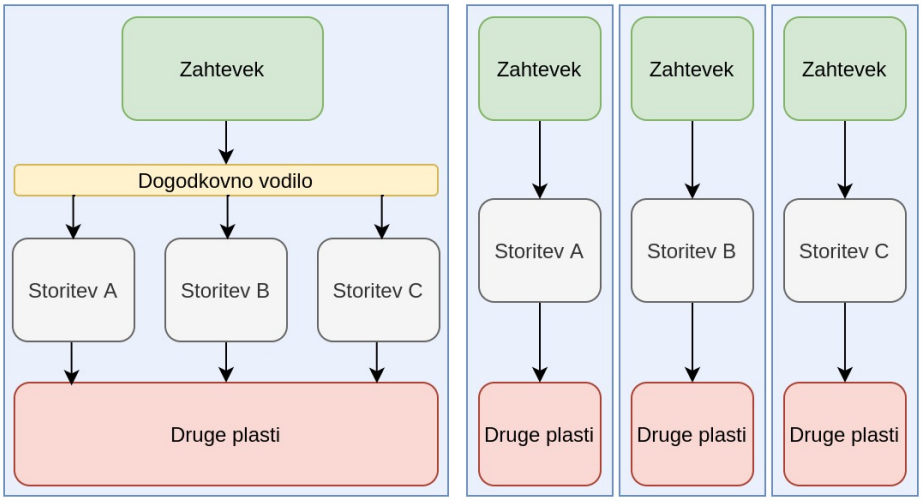
- preusmerjanje dogodkov iz začetne točke na poljubno končno točko;
- preoblikovanje dogodkov glede na zahteve končne točke;
- filtriranje in analiza dogodkov.

Dogodkovni prehod, o katerem govori diplomsko delo, deluje drugače od tipičnega dogodkovnega prehoda, saj nima neskončne zanke. To je mogoče, ker rešitev uporablja mikro storitev za zagon kode v oblaku, katero je mogoče nasloviti in pognati z unikatnim imenom. Tako lahko tehnologija, ki zaznava dogodek, direktno naslavlja kodo v oblaku.

3.2 Oblak

V preteklosti se je o strežnikih govorilo kot o kombinaciji strojne opreme, operacijskega sistema, shrambe in aplikacije. Ko so aplikacije na strežniku porabile vso kapaciteto, je bilo potrebno dodati nov strežnik, kar je bil dokaj zapleten proces. Takšno nadgrajevanje je pomenilo, da ponudba in povpraševanje skoraj nikoli nista bila v optimalnem razmerju. Prav tako je obstajala nevarnost za prenehanje delovanja strežnika, kar je lahko onesposobilo celotno storitev. Skupino strežnikov se lahko organizira v gručo oziroma porazdeljeni sistem (*ang.* distributed system), kar pomaga pri zanesljivosti, vendar takšna rešitev ni primerna za vse aplikacije. Korak bližje oblačnim storitvam so virtualni ali navidezni stroji, kjer je strojna oprema ločena od programske. Virtualni strežnik lahko gosti več operacijskih sistemov in aplikacij naenkrat, zaradi česar je tak strežnik stroškovno bolj učinkovit [10]. Mikro storitve so prav tako neodvisne aplikacije, ki si delijo strojno opremo, vendar so poleg tega ponujene kot storitev. To pomeni, da lahko podjetja uporabljajo mikro storitve in plačajo le toliko, kolikor porabijo. To je zelo

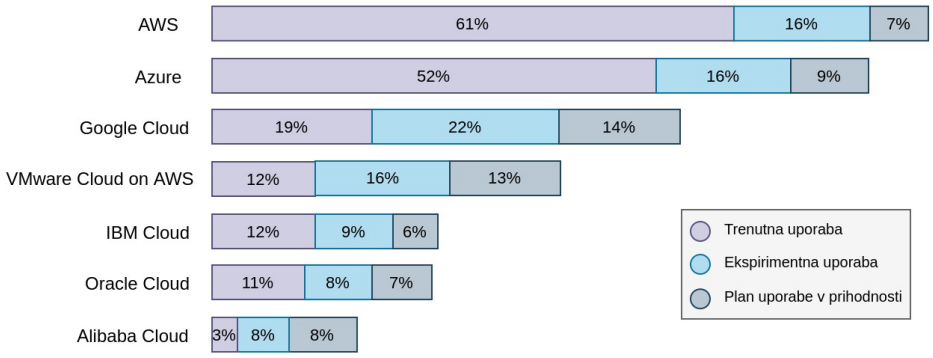
koristno za nova podjetja, saj v tem primeru niso potrebne investicije v drago opremo. Na sliki 3.1 lahko vidimo primerjavo storitveno umerjene arhitekture aplikacije z mikro storitveno strukturo aplikacije. V primerjavi opazimo, da storitveno usmerjena aplikacija sprejme zahtevek, katerega obdela dogodkovno vodilo in ga nato posreduje ustreznemu delu aplikacije. Komunikacija med vodilom in storitvami znotraj aplikacije je zato specifična za rešitev in ni prenosljiva na drugo aplikacijo. Pri mikro storitveno usmerjeni aplikaciji pa so storitve razdeljene na manjše dele, kjer vsaka mikro storitev prejme svoj zahtevek. Komunikacija med mikro storitvami je definirana kot API (določen naslov storitve ter vhod podatkov), zato je mogoče mikro storitve ponovno uporabiti v drugi aplikaciji. Druge plasti na sliki predstavljajo sredstva, ki jih storitev potrebuje za izvedbo, to vključuje vse od strojne opreme, shrambe, operacijskega sistema do okolja za izvajanje kode. V diplomskem delu uporabljamo takšno strukturo aplikacije, katero poskušamo optimizirati, zato je pomembno razumeti, kako deluje.



Slika 3.1: Storitveno usmerjena arhitektura v primerjavi z mikro storitveno arhitekturo.

3.2.1 Vodilni ponudniki oblaka

Glavni trije ponudniki oblaka danes so Amazon, Microsoft in Google, kjer je Amazon tisti, ki je prvi ponudil oblačne storitve. Uporaba oblaka že od samega začetka, ko je Amazon izdal prvo platformo leta 2002, hitro narašča, česar se vodilna podjetja zavedajo in vlagajo velike količine denarja v razvoj oblačnih storitev. Na sliki 3.2 lahko vidimo odgovore anketirancev glede uporabe oblačnih platform in načina uporabe platforme. V diplomskem delu se bomo ukvarjali s platformama AWS in Microsoft Azure, ki sta trenutno vodilni oblačni platformi in skupaj ponujata širok nabor storitev.



Slika 3.2: Vodilni ponudniki oblačnih storitev – 2019 [17].

Da si lahko bolje predstavljamo pomembnost oblaka in mikro storitev danes, si lahko ogledamo primere znanih programov, ki v svoji infrastrukturi uporabljajo oblačne storitve:

- distribucija video vsebin Netflix;
- spletna trgovina Amazon;
- podjetje za deljenje prevozov Uber;
- spletna tržnica Ebay;
- distribucija glasbenih vsebin Sound Cloud;
- ponudnik televizije, interneta, telefona Comcast Cable.

3.3 Mikro storitve

Mikro storitvene aplikacije so sestavljene iz več storitev, kjer ima vsaka storitev svoj določen API za komunikacijo z drugimi storitvami. Vsaka mikro storitev ponuja določeno funkcionalnost, ki teče v svojem procesu in je lahko enostavno povezana v modularno sestavljeno aplikacijo. Modularno sestavljanje aplikacij je koristno za hitrejši razvoj in razširljivost, saj so funkcionalnosti v oblaku že robustno podprte s strani oblačnih ponudnikov, ki jih v začetku ponujajo zastonj ali zelo poceni.

Pogosta praksa ponudnikov oblačne infrastrukture je, da prevzamejo poslovni model, ki omogoča brezplačno uporabo za manjše aplikacije, kar omogoča hitrejše pridobivanje novih uporabnikov. Kasneje, ko aplikacija začne nabirati več prometa, se poraba izračuna skladno z dejansko porabo. Taka vrsta zaračunavanja je koristna za nova podjetja, ki bi sicer morala porabiti velike količine denarja za domene in razvoj strežnikov [5]. Mikro storitve bomo v delu uporabili za sestavo preprostih aplikacij, kjer bomo omogočili komunikacijo med mikro storitvami različnih ponudnikov.

3.3.1 Osnovne storitve

Osnovne mikro storitve so storitve, ki so v očeh ponudnikov najbolj pomembne in uporabljene ter je ponudnik v njih vložil dodatno količino denarja in časa. Ponudniki se odločijo, katere storitve so bolj pomembne glede na povpraševanje uporabnikov in pokritost temeljnih funkcionalnosti. Pri razvoju tehnologije, kot je dogodkovni prehod, se je zato smiselno najprej osredotočiti na takšne storitve. V nadaljevanju bomo predstavili dve osnovni mikro storitvi, ki sta tudi uporabljeni v delu.

3.3.2 Funkcija kot storitev

Funkcija kot storitev (FaaS) (*ang.* function as a service) je način izvajanja kode v oblaku, kjer je okolje za izvedbo ponujeno kot storitev. Uporabnik lahko kodo naloži v oblak kot funkcijo in jo izvede ob določenem dogodku s

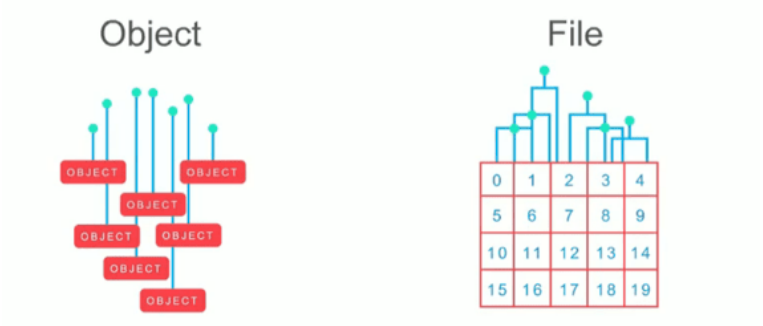
pomočjo aplikacijskega programskega vmesnika (*ang.* application programming interface (API)). Funkcija kot storitev pri večini oblačnih ponudnikov spada pod osnovne storitve, saj je zelo razširjena med uporabniki, pri ponudniku Amazon jo naslavljamo kot Lambda, pri ponudniku Azure pa kot funkcijsko aplikacijo (*ang.* FunctionApp). FaaS je za uporabnika koristna tehnologija, saj uporabnik ne rabi skrbeti za strežnike in njihove stroške, ampak lahko preprosto plačuje po uporabi funkcije. Funkcije v oblaku se zaračunajo glede na čas izvajanja in od podjetja ne zahtevajo administracije. Oblak poskrbi za pogon kode, razširljivost in dostopnost [9]. Funkcije naj bi bili manjši zaključeni deli kode, ki se izvedejo v nekem trenutku in opravijo določeno nalogo. Koda aplikacije je zato modularna v kontrastu z monolitnimi aplikacijami, kjer se vsa koda sproži naenkrat. Mikro storitev bomo v delu uporabili kot primer končne točke dogodkovnega prehoda in kot tehnologijo za prenos dogodka med ponudniki.

3.3.3 Objektna shramba

Objektna shramba (*ang.* bucket storage) je ena od osnovnih mikro storitev oblačnih ponudnikov, ki jo bomo uporabili v diplomskem delu za predstavitev rešitve dogodkovnega prehoda. To je specifično začetna točka prehoda, ki bo ob spremembi stanja datotek zaznala dogodek in ga nato posredovala dogodkovnemu prehodu. Da lahko bolje razumemo delovanje implementirane rešitve, si bomo mikro storitev v nadaljevanju podrobneje pogledali.

Objektna shramba se od tradicionalne shrambe z datotečnim sistemom razlikuje po tem, da ima zelo preprosto organizacijo, kjer ima vsak objekt enoličen ID za naslavljanje. Objektna hramba je hitro dostopna za aplikacije zaradi direktnega naslavljanja objektov preko tehnologije API in lahko hrani virtualno neskončno podatkov. Operacije nad objekti so preproste in omejene, kot na primer nalaganje datotek, prenos datotek, kopiranje datotek, brisanje datotek. Določene funkcionalnosti pa niso podprte, kot na primer iskanje, saj objektna shramba predpostavlja, da uporabnik že pozna unikatni ID objekta, do katerega želi dostopati [13]. Datoteke so lahko organizirane v

skupine, ki jim rečemo vedra (*ang.* bucket) in so namenjena za učinkovitejšo organizacijo datotek. Naletimo lahko tudi na drugačna poimenovanja, na primer zabojnik (*ang.* container) pri ponudniku Microsoft Azure. Na sliki 3.3.3 lahko vidimo primerjavo dostopa do datotek v objektni shrambi in v datotečni shrambi.



Slika 3.3: Primerjava objektne shrambe z datotečno shrambo [3].

Oblačni ponudniki pogosto ponujajo tudi obvestila o spremembah na vedru, kjer se lahko dogodek o spremembi usmeri na različne končne točke. Vrste dogodkov in končne točke sporočila se razlikujejo glede na ponudnika, vendar v osnovi delujejo na enak način. Za upravljanje z dogodki poskrbi ponudnik sam, uporabnik pa lahko nastavi pogoje dostavljanja.

3.4 Odvisnost od ponudnika

V diplomskem delu se pogosto srečamo z odvisnostjo od ponudnika, saj je to del problema, ki ga rešuje naloga. O odvisnosti od ponudnika govorimo, ko je strošek menjave ponudnika tako visok, da je stranka pod pritiskom, da ostane pri istem ponudniku [18]. Ko si izbere enega ponudnika, pa mora nadaljevati z uporabo storitev pri istem ponudniku, saj storitve niso kompatibilne s storitvami ostalih ponudnikov. Menjava ponudnika je sicer mogoča, vendar pomeni visoke stroške, za katere se uporabniki po navadi ne odločijo. Razlogov, zakaj bi uporabnik želel zamenjati ponudnika ali del storitev ponudnika, je veliko. Lahko gre za padec v kakovosti določene mikro storitve,

za veliko spremembo mikro storitve, ponudnik lahko spremeni ceno uporabe mikro storitve ali jo celo ukine. Prav tako pa bi si uporabnik morda želel združiti več ponudnikov v isti aplikaciji, saj ima vsak ponudnik svoje močne in šibke točke, in tako združiti močne točke ponudnikov, da doseže optimalno aplikacijo. To pa trenutno ni mogoče brez dodatnega vmesnika, ki omogoča kompatibilnost med mikro storitvami.

3.4.1 Kako se izogniti odvisnosti od ponudnika

Pri izbiri ponudnika podjetja uporabljajo različne strategije za preprečevanje odvisnosti, saj jim lahko napačna izbira povzroči velike stroške. Problem, ki ga rešuje diplomsko delo, je težavna komunikacija med mikro storitvami različnih ponudnikov, ki je tudi del problema odvisnosti od ponudnika, saj uporabnikom preprečuje enostavno izbiro poljubne mikro storitve. V nadaljevanju bomo naredili pregled pogostih strategij, ki jih podjetja uporabljajo za reševanje odvisnosti od ponudnika, da lahko bolje razumemo, kaj mora trenutno uporabnik narediti, da se izogne odvisnosti.

- Previdno ocenjeni ponudniki glede na potrebe podjetja in plan za zgodnji izhod
  Podjetja se pogosto zelo hitro odločijo za določenega ponudnika, brez da bi raziskala, kateri ponudnik je res najbolj primeren za njih in opravila postavitev testne aplikacije.

- Primerno zasnovane aplikacije za olajšano migracijo – ohlapno povezane aplikacije (*ang.* loosly coupled)
  Če podjetje poskrbi, da so aplikacije prenosljive, jim to omogoča lažji prenos. Podjetje lahko zagotovi, da je aplikacija bolj prenosljiva tako, da dobro definira podatkovne modele in podatke hrani v formatih, ki so uporabljeni pri več oblačnih ponudnikih. Prav tako lahko poskrbi, da je aplikacija ohlapno povezana in s tem bolj prenosljiva. Podjetje lahko to doseže z uporabo tehnologij, kot so REST API, HTTP, JSON ...

- Varnostne kopije podatkov
  Kopije podatkov so koristne, saj je ekstrakcija vseh podatkov iz oblaka lahko časovno zahtevna in draga.

- Uporaba več-oblačnih ali hibridnih rešitev in strategij
  Podjetje lahko namesto storitve uporabi zasebno storitev ali storitev drugega ponudnika, če to trenutni oblak omogoča. V to strategijo reševanja odvisnosti od ponudnika spada tudi dogodkovni prehod, o katerem govori diplomsko delo, saj uporabniku omogoča uporabo več-oblačnih rešitev, kljub temu, da sami ponudniki tega ne omogočajo.

- Implementacija DevOps procesov in orodij
  Pri razvoju DevOps procesov lahko podjetje zagotovi neodvisnost posameznih delov programske opreme z orodji, ki omogočajo izolacijo programske opreme od okolja in tako zagotovijo prenosljivost programske opreme. Prav tako je koristna avtomatska konfiguracija okolja s pomočjo orodij za avtomatizacijo in orkestracijo. Primer takšne konfiguracije sta Ansible in xOpera skripti, ki sta predstavljeni v nadaljevanju [1].

V primeru, da podjetje kljub previdnosti naleti na problem, kjer ga omejuje odvisnost od ponudnika, lahko v določenih primerih uporabi dogodkovni prehod in se tako izogne velikim stroškom ali uporabi manj primerne mikro storitve.

Poglavje 4

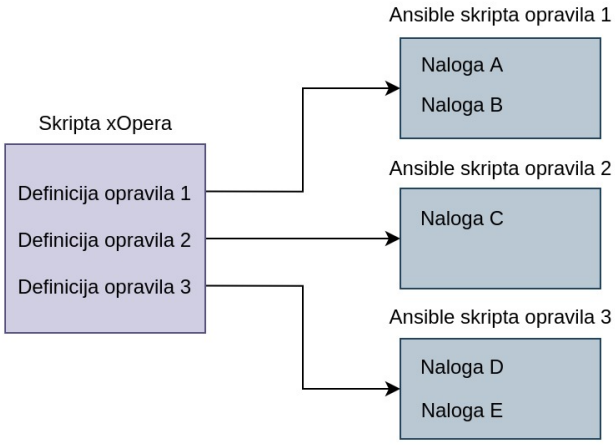
Razvoj rešitve

4.1 Uporabljene tehnologije in orodja

Za razvoj kakovostnega orodja je pomembna izbira ustreznih tehnologij. V tem poglavju bomo predstavili glavne tehnologije, ki so bile uporabljene med razvojem dogodkovnega prehoda.

Na sliki 4.1 je predstavljen način uporabe tehnologij xOpera in Ansible pri snovanju aplikacije. Levo vidimo xOpera skript, ki vsebuje definicije opravil v aplikaciji. Definiciji opravila lahko pripnemo implementacijo opravila, ki je logična skupina manjših nalog, katere izvaja Ansible. V diplomskem delu je takšna struktura aplikacije uporabljena v fazi snovanja in raziskovanja rešitve zaradi preproste sintakse programskih jezikov in ogrodij ter preglednosti strukture rešitve.

Kasneje, ko prenehamo s fazo raziskovanja in testiranja možnih rešitev, lahko razvijemo aplikacijo, ki je za uporabnika bolj intuitivna. Odjemalni program v terminalu je dober nadomestek uporabniškega vmesnika, saj lahko uporabnik interaktivno vnaša parametre in je enostaven za namestitev. Program razvijemo v programskem jeziku Python.



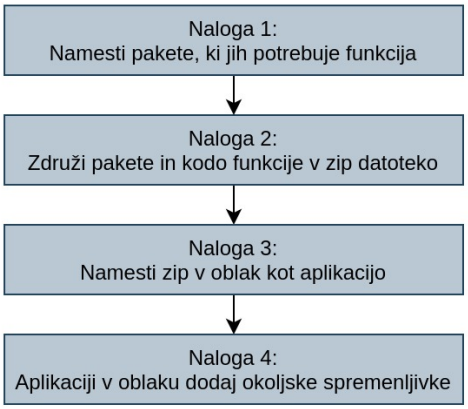
Slika 4.1: Prikaz uporabe tehnologij xOpera in Ansible pri sestavi aplikacije.

4.1.1 Ansible

Ansible je odprtokodno orodje za avtomatizacijo, ki je primarno namenjeno nalaganju aplikacij, nadgrajevanju delovnih postaj in strežnikov ter konfiguraciji in zagotavljanju stanj programske opreme. Ansible skript podani sistem ali skupino sistemov konfigurira in jih postavi v določena stanja [19]. Skripti so napisani v jeziku YAML, ki je zasnovan za lahko berljivost in razumevanje.

Tehnologija Ansible je v tem diplomskem delu uporabljena za prikaz konfiguriranja in nalaganja aplikacije v oblak ter za ponovljivo uporabo aplikacije na različnih uporabniških računih v oblaku. V Ansible skriptu najdemo skupek nalog, ki sestavljajo določeno opravilo. Na sliki 4.2 vidimo primer opravila za sestavljanje in nalaganje funkcije v oblak. Vsaka naloga prejme parametre, ki so definirani v dokumentaciji Ansible modulov in jih nalogi poda uporabnik.

Ansible skripti bodo poskrbeli za pripravo mikro storitev, vključno s konfiguracijo zaznavanja spremembe na vedru in pripravo kode funkcije, ter bodo poskrbeli za stanje aplikacije pred uporabo dogodkovnega prehoda.



Slika 4.2: Naloge v ansible skriptu za postavitev funkcije v oblaku.

4.1.2 xOpera

Je odprtokodno orodje za orkestracijo aplikacij v oblaku, razvito skladno s TOSCA OASIS ogrodjem. Uporablja se za konfiguracijo vozlišč in povezavo med njimi ter omogoča orkestracijo celostne aplikacije. Definicija vozlišča vsebuje pot do Ansible skripta, ki poskrbi za postavitev vozlišča na oblak. Skript zagotavlja, da se aplikacija postavi v opisano stanje, kar pomeni, da se deli aplikacije, ki že obstajajo, ne postavijo ponovno. V diplomskem delu xOpera skripte uporabljamo za predstavitev aplikacije uporabnika in za snovanje dogodkovnega prehoda za minimalni primer [20]. Ker skript opisuje stanje aplikacije, ga lahko uporabimo za postavitev aplikacije na različnih računih ali za zagotavljanje stanja aplikacije na že obstoječem računu.

4.1.3 Python

Python je visoko stopenjski in več namenski objektni programski jezik. Zasnovan je bil za dobro berljivost in hitro pisanje kode. V diplomskem delu uporabljamo Python za hitro snovanje in testiranje funkcij in za razvoj odjemalnega programa.

Odjemalni program si želimo napisati v programskem jeziku Python tudi

zato, da ga lahko zapakiramo kot paket in ga shranimo v PyPI repozitorij. PyPI (*ang.* python package index) je repozitorij ali zbirka za programsko opremo v jeziku Python, ki uporabnikom omogoča hitro in enostavno pridobivanje programske opreme glede na želeno verzijo. Program pip uporablja PyPI repozitorij kot privzet vir za pridobivanje programske opreme [16], kar uporabniku olajša nalaganje paketov, ki jih potrebuje za svoje Python aplikacije. Z definiranjem uporabljenih verzij programov lahko zagotovimo delovanje aplikacije ne glede na nadaljnji razvoj programov.

4.1.4 Postman

Programska oprema Postman vsebuje več funkcionalnosti in orodij. V diplomskem delu je specifično uporabljen odjemalni program Postman API, ki uporabniku omogoča hitro sestavo zahtevkov in testiranje API tehnologij. Orodje omogoča hitro nastavitev parametrov s pomočjo jasnega uporabniškega vmesnika, pošiljanje zahtevka in pregled odgovorov zahtevka. Uporabnik lahko po vnosu parametrov tudi zgenerira kodo za pošiljanje zahtevka v različnih programskih jezikih. Orodje v diplomskem delu uporabljamo za naslavljanje mikro storitve `API gateway` [15], kateri pošljemo zahtevek, ki ga zazna poslušalec dogodkov.

4.2 Izbira ponudnikov

Za prikaz rešitve dogodkovnega prehoda izberemo dve vodilni platformi za oblačne storitve, AWS in Microsoft Azure, med katerima bi bil dogodkovni prehod zelo zaželen. Z dogodkovnim prehodom bomo omogočili uporabo mikro storitev obeh ponudnikov v isti aplikaciji. Kasneje se lahko dogodkovni prehod razširi še na druge aktualne ponudnike oblaka.

4.2.1 AWS

Ponudnik AWS (*ang.* Amazon web services) je trenutno vodilni ponudnik oblačnih storitev. Ideja za AWS se je začela leta 2000, ko je bilo podjetje Amazon le ponudnik elektronskega nakupovanja izdelkov. Podjetje Amazon je imelo zaradi svoje uspešnosti problem z razširljivostjo storitve, zato so vložili veliko denarja v nakup kakovostnih sistemov. To je vodilo do ideje, da bi se takšne sisteme ponudilo uporabnikom kot storitev. Ponudnik AWS je v začetku naletel na velike težave pri snovanju sistema, ki ga lahko ponudijo podjetjem kot storitev, zato so zaposlili več novih inženirjev in zasnovali skupino API vmesnikov, ki povezuje danes tako imenovane mikro storitve [8].

Platforma AWS za komunikacijo in naslavljanje med mikro storitvami uporablja posebno ime ARN (*ang.* Amazon resource name), ki je sestavljeno iz informacij o viru in je unikatno. Unikatno ime je pomembno, saj komunikacija poteka preko tehnologije REST API. ARN ali unikatno ime vira bomo uporabljali v nadaljevanju dela za komunikacijo med mikro storitvami.

Struktura ARN:

```
arn:partition:service:region:acc-id:resource-type:resource-id
```

V nadaljevanju dela bomo, ko govorimo o ponudniku AWS, objektno shrambo naslavljali kot AWS S3 (*ang.* Simple Storage Service), funkcijo v oblaku pa kot AWS Lambda.

4.2.2 Microsoft Azure

Danes je Microsoft Azure drugi vodilni ponudnik oblačnih storitev, zato si želimo raziskati, kako ga lahko povežemo z AWS. Microsoft je napovedal platformo Azure dve leti po tem, ko je AWS izdal svoj produkt „infrastruktura kot storitev“. Produkt Azure pa je izšel leta 2010 in je dobil stroge kritike, saj so ga primerjali z AWS. Od takrat je Azure dohitel konkurenco in zaseda svoj del trga s kakovostnimi storitvami.

Platforma Microsoft Azure za komunikacijo med mikro storitvami prav tako uporablja tehnologijo REST API, vendar kot unikatno ime vira uporablja klasičen URL (*ang.* Uniform Resource Locator).

Struktura URL:

```
http://<APP_NAME>.azurewebsites.net/api/<FUNC_NAME>?code=key
```

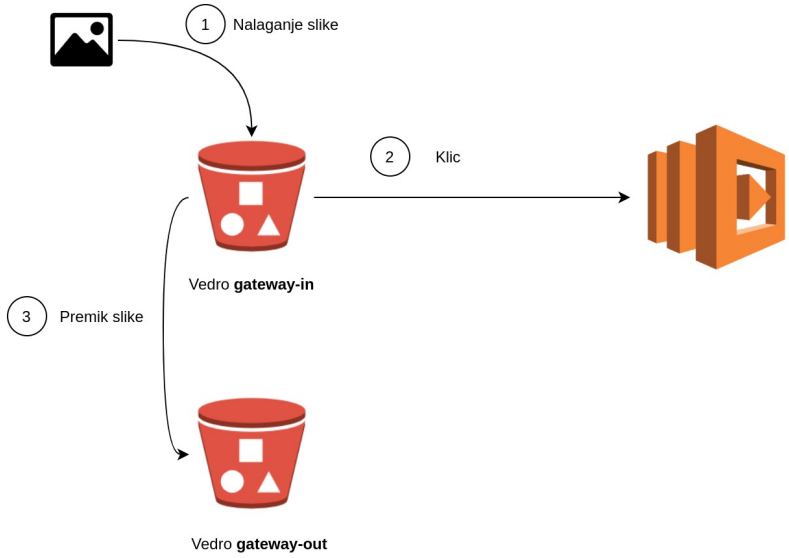
V nadaljevanju, ko govorimo o Microsoft Azure, objektno shrambo naslavljamo kot Microsoft Azure zabojniki (*ang.* containers), funkcijo pa kot Microsoft Azure funkcija (*ang.* function).

4.3 Uporaba mikro storitev znotraj ponudnika

V temu razdelku bomo predstavili implementirano aplikacijo uporabnika pred potrebo po menjavi ponudnika oblaka oziroma določene mikro storitve ponudnika. To je pomembno zato, da vidimo, kakšna je aplikacija, če uporabljamo storitve enega ponudnika. Kasneje bomo ta primer uporabili za prikaz dogodkovnega prehoda med ponudniki in tako videli razliko med rešitvama. Aplikacija, ki jo bomo implementirali, deluje tako, da sliko s pomočjo funkcije prenese iz ene shrambe v drugo. Takšna aplikacija je koristna za obdelovanje slik, saj dandanes obstaja veliko število zaslonov različnih velikosti, resolucij in podobno, ki imajo različne zahteve za slike. Aplikacija za spreminjanje velikosti slik je pogosto uporabljena za primere in demonstracije, saj združuje dve osnovni mikro storitvi in je kljub preprostosti dober primer uporabnosti aplikacij v oblaku. Ker, v našem primeru, želimo čim bolj preprost primer aplikacije, slike med prenosom ne bomo spreminjali.

Primer aplikacije v oblaku uporablja dve mikro storitvi, objektno shrambo (omenjeno v razdelku 3.3.3) in funkcijo (omenjeno v razdelku 3.3.2). V objektni shrambi definiramo dve vedri in ju poimenujemo `gateway-in` in `gateway-out`. Na vedru lahko nastavimo zaznavanje različnih dogodkov, kot

so dodajanje objekta, brisanje objekta, obnovitev objekta itd. V našem primeru definiramo zaznavanje dogodka za nov dodan objekt s končnico „.jpg“. Ob prenosu datoteke v vedro `gateway-in` se sproži dogodek, ki ga lahko naslovimo na več različnih končnih točk. Končne točke se razlikujejo glede na ponudnika oblaka in bodo opisane v nadaljevanju dela. Pri obeh ponudnikih imamo možnost povezave obvestila na funkcijo znotraj ponudnika. Obvestilo bo sprožilo funkcijo, ki bo izvedla določeno akcijo, na primer prenos datoteke iz vedra `gateway-in` v vedro `gateway-out`. Potek akcij opisane aplikacije je prikazan na sliki 4.3.



Slika 4.3: Potek akcij v osnovni aplikaciji, ki je sestavljena iz mikro storitev istega ponudnika.

Predpostavljeno aplikacijo je mogoče pripraviti izključno s pomočjo platforme ponudnika, vendar jo za potrebe testiranja in ponovljivosti zapišemo v skript. Skript je napisan po standardu TOSCA in vsebuje definicije ločenih delov oziroma nalog aplikacije ter opisuje odnose med definiranimi nalogami. Standard TOSCA smo že omenili v poglavju 4.1.2, kjer smo povedali, za

kaj služi v kombinaciji z orodjem xOpera, ki omogoča postavitev aplikacije, napisane v standardu TOSCA. Naloge so razdeljene na več skriptov, ki se imenujejo (*ang.* ansible playbooks) in na katere se sklicujemo v TOSCA skriptu pod poljem `implementation`. TOSCA skript zaženemo s pomočjo programa xOpera, ki poskrbi za razčlenitev in pogon skripta z uporabnikovimi parametri.

Za osnovno aplikacijo uporabnika napišemo več nalog, npr. AWS S3, AWS lambda, Microsoft Azure function, Microsoft Azure container ... Vsak skript naloge zagotavlja opisano stanje na oblačnem strežniku. V primeru AWS lambde to pomeni pripravo funkcije, nalaganje funkcije, nalaganje okoljskih spremenljivk funkcije, konfiguracijo dovoljenj in podobno.

Za prikaz prehoda pripravimo dve osnovni aplikacij. Prva pripravi objektno shrambo z dvema vedroma `gateway-in` in `gateway-out` na platformi AWS in funkcijo na platformi Microsoft Azure, druga aplikacija pa pripravi objektno shrambo z dvema vedroma na Microsoft Azure in funkcijo na AWS. V nadaljevanju dela bo predpostavljeno, da sta ti aplikaciji že postavljeni na oblaku.

4.4 Minimalni primer uporabe dogodkovnega prehoda

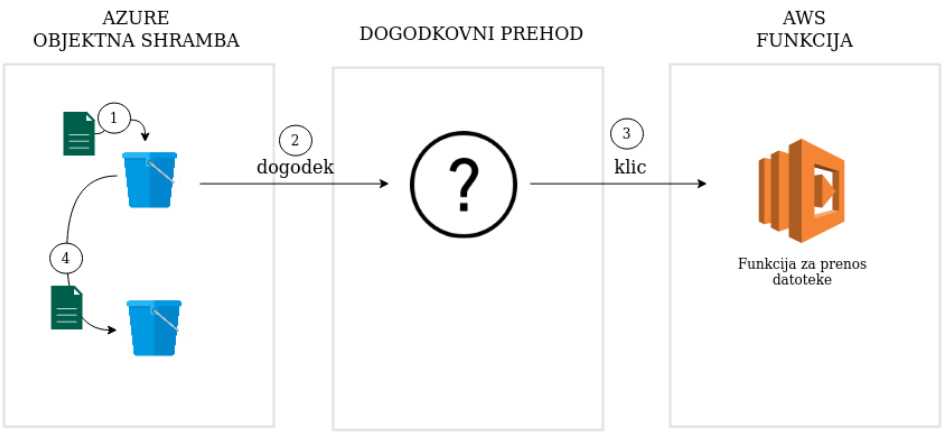
Rešitev dogodkovnega prehoda bo najprej testirana z minimalnim primerom uporabe, kjer bo sproženi dogodek preusmerjen iz enega ponudnika na drugega. Za primer bo uporabljena objektna shramba, ki bo ob nalaganju slike sprožila dogodek. Dogodek se bo nato prenesel na dogodkovni prehod. Od tam bo prehod poklical funkcijo drugega ponudnika, ki bo sliko iz vedra `gateway-in` prestavila v vedro `gateway-out`. Omenjena aplikacija je že opisana v razdelku 4.3.

Uporabnik, ki uporablja dva različna oblačna ponudnika Microsoft Azure in AWS ter ima pri vsakemu po eno mikro storitev (objektno hrambo ali

funkcijo kot storitev), lahko uporabi dogodkovni prehod za konfiguracijo komunikacije med njima oziroma mikro storitvi poveže tako, da je objektna shramba zmožna poslati obvestilo funkciji. Predpogoj za uporabo dogodkovnega prehoda sta že postavljeni začetna in končna točka prehoda. Uporabnik z ukazom `pip install event-gateway-cli` namesti program za konfiguracijo dogodkovnega prehoda in mu poda informacije o začetni in končni točki.

4.4.1 Povezava AWS lambda in Microsoft Azure container

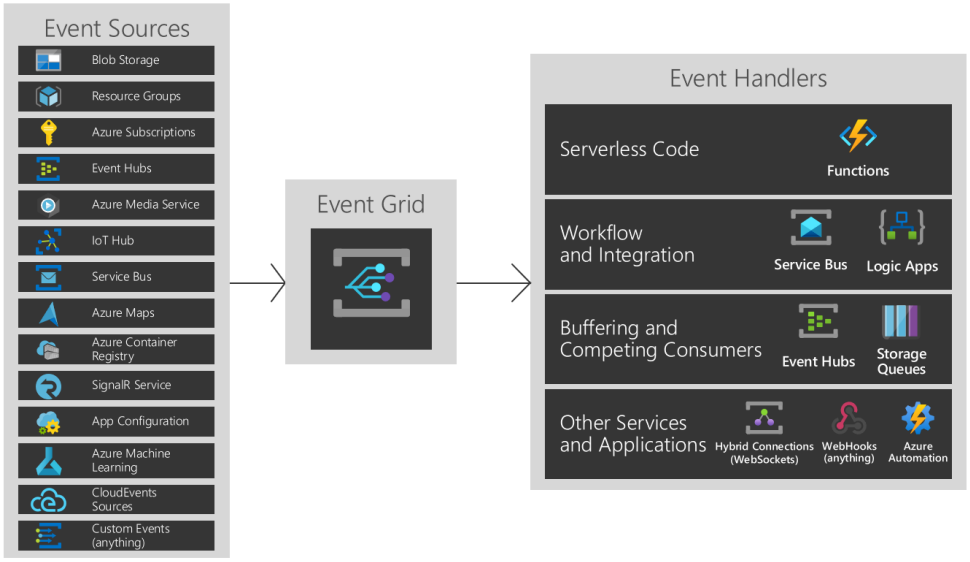
Pri raziskavi rešitve dogodkovnega prehoda za primer najprej vzamemo eno od aplikacij, omenjenih v poglavju 4.3. Predpostavljamo, da sta začetna in končna točka prehoda že postavljeni, kot prikazano na sliki 4.4. Razdelek `dogodkovni prehod` predstavlja manjkajoči del pri povezavi AWS lambda in Microsoft Azure container, implementacija katerega bo predstavljena v nadaljevanju. Puščici `2` in `3` predstavljata vhod in izhod dogodkovnega prehoda, ki ju moramo upoštevati.



Slika 4.4: Grafični prikaz strukture aplikacije za prenos dogodka iz Microsoft Azure na AWS, ki bo razvita v nadaljevanju.

Končne točke za Microsoft Azure container obvestila

Najprej moramo analizirati začetno točko prehoda in ugotoviti, kaj omogoča glede pošiljanja dogodkov. Azure za prepošiljanje obvestil uporablja svojo storitev, ki se imenuje obvestilna mreža (*ang.* event grid). O tehnologiji event grid smo že govorili v razdelku 2.2. Microsoft Azure je glede omogočanja prepošiljanja obvestil izven ponudnika napreden, saj omogoča veliko končnih točk, med katerimi so tudi takšne, ki omogočajo preusmerjanje na druge aplikacije izven Microsoft Azure oblaka. Microsoft Azure EventGrid je tudi neke vrste dogodkovni prehod, vendar se ne osredotoča na povezljivost med različnimi ponudniki oblaka in njihovimi mikro storitvami. Na sliki 4.5 lahko v razdelku „event sources“ vidimo Azure mikro storitve, ki so zmožne dogodke poslati na event grid prehod, in v razdelku „event handlers“ končne točke, na katere lahko tehnologija event grid naslovi dogodke.



Slika 4.5: Microsoft Azure Event grid shema za rokovanje z dogodki [2].

Končne točke EventGrid obvestil so:

- Webhooks ali spletne kljuke,
  To je tehnologija za učinkovito pošiljanje podatkov aplikacijam, ki zahteva konfiguracijo na unikatnem URL naslovu. Koristna je za rokovanje z in proženje dogodkov, brez uporabe velike količine virov (*ang.* resource)
- Microsoft Azure functions ali Microsoft Azure funkcije;
- Event hubs ali dogodkovno vozlišče;
- Hybrid connections ali hibridne povezave;
- Service Bus queues and topics ali čakalne vrste in teme;
- Storage queues ali čakalne vrste za dogodke shrambe.

Za potrebe dogodkovnega prehoda med oblačnimi ponudniki nas najbolj zanimajo končne točke iz zadnjega razdelka na sliki 4.5 „Druge storitve in aplikacije“ (*ang.* Other Services and Applications). Izmed naslednjih nam najbolj ustreza webhook tehnologija, saj deluje preko HTTP zahtevkov, kar pomeni, da lahko na preprost način preusmerimo obvestilo na dogodkovni prehod. Uporabnik bi v tem primeru na prvi pogled lahko vzpostavil direktno komunikacijo med Microsoft Azure EventGrid in AWS lambdo, vendar se stvari zapletejo zaradi varnostnih mehanizmov na obeh straneh komunikacije. Microsoft Azure webhook tehnologija zahteva ob prvi poizvedbi potrditveni odgovor z žetonom, po katerem pa komunikacija deluje normalno. Na strani AWS lambde pa je potrebna konfiguracija API prehoda, ki omogoča naslavljanje lambde iz zunaj oblaka, vendar takšno naslavljanje API prehoda zahteva generiranje podpisa. Oba varnostna koraka bi uporabniku močno otežila pisanje funkcij, zato je tudi v tem primeru smiselno implementirati dogodkovni prehod med ponudniki oblaka.

Razvoj funkcije za prenos dogodka

Funkcija za prenos dogodka med Microsoft Azure objektno shrambo in AWS funkcijo se mora nahajati pri ponudniku Microsoft Azure. To nam omogoča

kompatibilnost med začetno točko Microsoft Azure objektne shrambe in funkcijo dogodkovnega prehoda. Funkcija se lahko nahaja na istem uporabniškem računu kot začetna in končna točka, ali pa na tujem uporabniškem računu, kjer je ponujena kot storitev. Dogodek, ki ga prejme funkcija dogodkovnega prehoda, je v JSON formatu in je definiran s strani Microsoft Azure ponudnika. Pri ponudniku Microsoft Azure je prvi klic, ob konfiguraciji poslušalca dogodkov na funkcijo, potrdilni klic, zato mora funkcija odgovoriti z žetonom. Naslednji klic na funkcijo se zgodi ob zaznanem dogodku na začetni točki in je preusmerjen na končno točko. V primeru, da je končna točka AWS funkcija, je od uporabnika zahtevano, da pripravi še AWS API gateway ali API prehod, ki omogoča klic funkcije izven ponudnika. Funkcija dogodkovnega prehoda zato ne naslavlja AWS funkcije direktno, temveč preko AWS API prehoda. Podatke o končni točki funkciji posredujemo preko okoljskih spremenljivk, kot prikazano spodaj.

```
aws_access_key = os.getenv('AWS_ACCESS_KEY_ID')
aws_secret_key = os.getenv('AWS_SECRET_ACCESS_KEY')
host = os.getenv('API_HOST')
```

Proces priprave in postavitve funkcije je zapisan v xOpera skriptu `gateway_container_to_lambda.yml`, ki vsebuje definicije za vozlišča `function_deploy` in `container_notification`. Vozlišče `function_deploy` poskrbi za postavitev funkcije in objektne shrambe, kamor se shrani koda funkcije. V tem koraku nastavimo tudi okoljske spremenljivke funkcije, ki vsebujejo konfiguracijske podatke, katere poda uporabnik. Nazadnje pa v skriptu `container_notification` nastavimo še poslušalca dogodkov, kjer podamo funkcijo dogodkovnega prehoda kot končno točko.

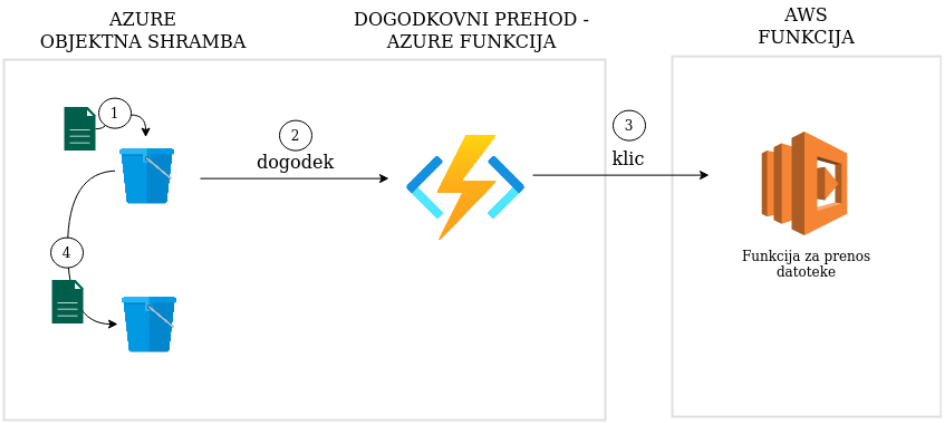
Uporabnik parametre, ki jih potrebuje orkestrator, poda v datoteko `inputs_container_to_lambda.yml` in jo referencira v ukazu, ki omogoča postavitev dogodkovnega prehoda, kot je prikazano spodaj.

```
$ opera deploy -i inputs_container_to_lambda.yml \\
gateway_container_to_lambda.yml
```

Testiranje

Kot test, če aplikacija pravilno deluje, odpremo objektno shrambo (*ang.* storage account) na Microsoft Azure oblak platformi in v vedro `gateway-in` naložimo sliko `slika.jpg`. Če je bil prenos dogodka uspešen, se bo tudi v `gateway-out` prikazala `slika.jpg`. Če prenos dogodka ni bil uspešen, lahko pogledamo v izpise za spremljanje funkcij, če so bile poklicane in kaj je šlo narobe. Testiranje in odpravljanje napak v kodi v oblaku je precej zahtevno, saj lokalno testiranje ni vedno mogoče, zato je treba aplikacijo vsakič znova postaviti.

Na sliki 4.6 vidimo strukturo celotne aplikacije skupaj z dogodkovnim prehodom za prenos dogodka iz platforme Microsoft Azure na AWS.

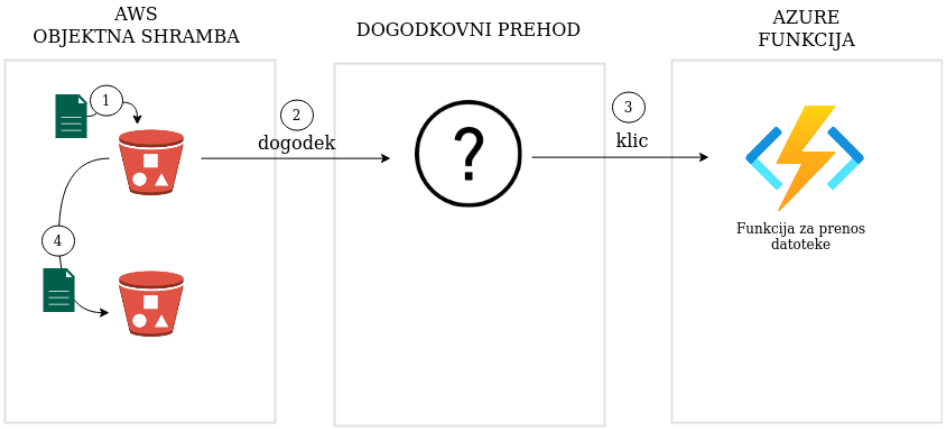


Slika 4.6: Grafični prikaz strukture aplikacije in dogodkovnega prehoda za prenos dogodka med Microsoft Azure in AWS.

4.4.2 Povezava AWS S3 bucket in Microsoft Azure function

Pregled možnih rešitev naredimo še za drugo od aplikacij, omenjenih v poglavju 4.3. Predpostavljamo, da sta začetna in končna točka prehoda že postavljeni, kot prikazano na sliki 4.7. Razdelek `dogodkovni prehod` pred-

stavlja manjkajoči del pri povezavi AWS S3 bucket in Microsoft Azure function, implementacija katerega bo predstavljena v nadaljevanju. Puščici 2 in 3 predstavljata vhod in izhod dogodkovnega prehoda, ki ju moramo upoštevati.



Slika 4.7: Grafični prikaz strukture aplikacije za prenos dogodka med AWS in Microsoft Azure, ki bo razvita v nadaljevanju.

Končne točke AWS S3 obvestila

Obvestilo ob spremembi v vedru ima določen nabor možnih končnih točk, ki so namenjene preusmerjanju dogodka znotraj ponudnika. Te končne točke pri AWS S3 so naslednje:

- AWS Simple Notification Service (SNS) ali storitev za preprosta obvestila;
- AWS Simple Queue Service (SQS) ali storitev za preprosto vrsto;
- AWS Lambda.

Vsaka od zgornjih končnih točk bi lahko bila uporabljena kot od uporabnika neodvisna storitev za preusmerjanje obvestila izven ponudnika AWS. Vendar sta za implementacijo bolj primerni storitvi SNS in AWS lambda, saj omogočata prenos dogodka brez aktivnega sodelovanja končne funkcije.

V nadaljevanju bomo naredili kratek pregled storitev SNS in AWS lambda in se odločili za tisto, ki je bolj primerna.

AWS SNS

SNS, storitev za preprosta obvestila (*ang.* simple notification service), deluje na principu tem. Uporabnik definira temo ali skupino, na katero objavlja sporočila, katera so nato preposlana velikemu številu prejemnikov. Ime teme ali skupine je lahko sestavljeno iz več nivojev, ki so med sabo ločeni z „/“. Princip teme je koristen in fleksibilen zaradi možnosti naslavljanja in filtriranja po skupinah.

Pri AWS SNS za testni primer definiramo temo `azure`, ki bo prejela obvestila od AWS S3 vedra in obvestilo nato preusmerila na Microsoft Azure. Da dosežemo prenos sporočila, je potrebno definirati SNS temo in naročnika na temo. Pri definiranju teme nas najbolj zanima ime in dostopna politika (ang *access policy*), ki mora dovoljevali dostop pošiljatelja dogodka do teme, v našem minimalnem primeru uporabe je to S3 vedro.

```
{
      "Sid": "s3",
      "Effect": "Allow",
      "Principal": {
        "Service": "s3.amazonaws.com"
      },
      "Action": "SNS:Publish",
      "Resource": "arn:aws:sns:eu-central-1:582:Azure"
}
```

Ko je dostopna politika definirana in omogoča povezavo z AWS S3 vedrom, se lahko ta poveže na temo preko sprožilca za dogodke. Težava pri storitvi SNS nastopi, ko želimo, da dogodkovni prehod uporablja več uporabnikov. Če je storitev SNS javna, mora program za konfiguracijo dogodkovnega prehoda vsakemu uporabniku nastaviti temo in naročnino, kar pa nasprotuje namenu

tem. Prav tako dogodkovni prehod ne bi bil sposoben nastavitve konfiguracije za novega uporabnika, brez da prekine delovanje že nastavljenih konfiguracij.

AWS lambda

Lambda ali funkcija kot storitev je kos kode, ki ga je mogoče sprožiti s pomočjo tehnologije API. Pregled funkcije kot storitve smo že obdelali v poglavju pregled tematike 3.3.2.

Funkcija bi bila zelo primerna za dogodkovni prehod, saj ponudniki večini mikro storitev omogočajo preusmeritev na funkcijo. To bi pomenilo, da bi bil dogodkovni prehod razširljiv tudi na druge mikro storitve ponudnika, ki niso omenjene v tem delu.

Funkcijo je mogoče konfigurirati na način, da dovoli dostop le določenim storitvam znotraj določenega uporabnika. Dogodkovni prehod bi zato lahko vsakemu uporabniku postavil funkcijo za prenos obvestil, ki bi lahko delovala tudi za več kombinacij storitev in bi bila dostopna le želenim uporabnikom.

Razvoj funkcije za prenos dogodka

Pri razvoju funkcije za preusmeritev dogodka moramo upoštevati že določene vhodne podatke, ki sestavljajo dogodek, in varnostne mehanizme na obeh straneh komunikacije.

AWS omogoča varen dostop do funkcij znotraj ponudnika s pomočjo dokumenta s politiko delovanja (*ang.* policy). Policy dokument vsebuje parametre za omogočanje dostopa glede na uporabniški račun, storitev, ki dostopa, akcijo, ki jo lahko izvede uporabnik, in podobno. Dokument se naloži na strežnik s posebnim ukazom in ni del kode funkcije.

Microsoft Azure pa osnovno varnost funkcije zagotavlja z URL za klic funkcije, ki vsebuje ključ. Vsak uporabnik, ki pozna URL in ključ, lahko pokliče funkcijo. Možne so tudi dodatne varnostne nastavitve, ki varujejo funkcijo pred napadi. Takšne dodatne nastavitve bi morale biti dodane v konfiguracijo dogodkovnega prehoda kasneje.

Funkcijo konfiguriramo tako, da na vhodu prejme JSON objekt z informacijami o dogodku. Objekt je avtomatsko zgeneriran s strani S3 vedra in ga ne moramo spremeniti. Nato s pomočjo python knjižnice `requests` sestavimo klic na Microsoft Azure funkcijo oziroma končno točko prehoda. Klic mora vsebovati URL za dostop in podatke, ki jih želimo posredovati. Parametre kot je URL za dostop, mora funkciji podati uporabnik dogodkovnega prehoda. Takšne parametre funkciji podamo preko okoljskih spremenljivk, kot prikazano spodaj. Na ta način tudi poskrbimo za varnost, saj občutljivi parametri v kodi niso vidni.

```
endpoint = os.getenv('AZURE_FUNCTION_ENDPOINT')
```

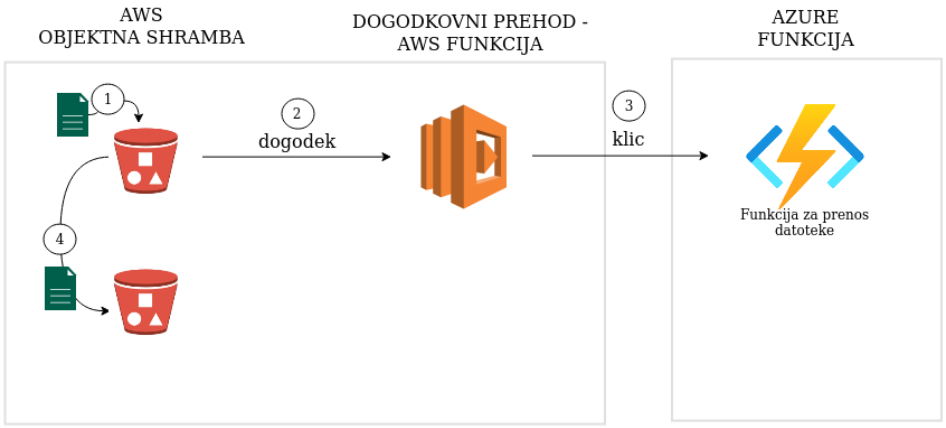
Proces priprave in postavitve funkcije je zapisan v xOpera skriptu `gateway_s3_to_function.yml`, kjer skript vsebuje definicije za vozlišča `aws_lambda_role`, `aws_lambda`, `bucket_notification`. Vozlišče `aws_lambda_role` je identiteta, ki določa dovoljenja uporabnika. Ko na oblak objavimo funkcijo, ji moramo določiti takšno identiteto z dovoljenji, da lahko izvaja svoje akcije. Vozlišče `aws_lambda` poskrbi za pripravo paketov, ki so uporabljeni v kodi, nato funkcijo stisne v `zip` datoteko in jo skupaj z okoljskimi spremenljivkami naloži na oblak. Da bo začetna točka prehoda lahko dostopala do funkcije, moramo omogočiti dostop za določeno mikro storitev in ID uporabniškega računa. Zadnji korak pa je konfiguracija vozlišča `bucket_notification` ali poslušalca dogodkov, ki na začetni točki zazna dogodek dodajanja „.jpg“ slike v objektno shrambo. Uporabnik parametre, ki jih potrebuje orkestrator, poda v datoteko `inputsS3_to_function.yml` in se nanjo sklicuje v ukazu za postavitev dogodkovnega prehoda, kot je prikazano spodaj.

```
$ opera deploy -i inputs_s3_to_function.yml \\
gateway_s3_to_function.yml
```

Testiranje

Za test pripravljene aplikacije na platformi AWS odpremo objektno shrambo (*ang.* storage account) in v vedro `gateway-in` naložimo sliko `slika.jpg`. Če je bil prenos dogodka uspešen, se bo `slika.jpg` prikazala tudi v `gateway-out`. Odpravljanje težav v kodi je precej pospešeno s sprotnim izpisovanjem uporabljenih parametrov, ki jih lahko spremljamo pod razdelkom (*ang.* monitoring) na platformi ponudnika oblaka.

Na sliki 4.8 lahko vidimo celotno aplikacijo skupaj z dogodkovnim prehodom za prenos dogodka med platformo AWS in Microsoft Azure.



Slika 4.8: Grafični prikaz strukture aplikacije in dogodkovnega prehoda za prenos dogodka med AWS in Microsoft Azure.

4.4.3 Odjemalni program za konfiguracijo dogodkovnega prehoda

Uporabniku je omogočena konfiguracija dogodkovnega prehoda s pomočjo odjemalnega programa v terminalu. Program od uporabnika zahteva parametre, kot so začetni in končni ponudnik ter začetna in končna mikro storitev.

```
Source provider: AWS
Source microservice: S3
```

```
Destination provider: Azure
Destination microservice: Function
```

Program s pomočjo knjižnic, ki jih ponujajo ponudniki oblaka, nastavi dogodkovni prehod za uporabnikove potrebe in tako uporabniku prihrani veliko dela. Program je napisan v programskem jeziku Python in je pripravljen za potencialno objavo kot PyPi paket, kjer ga uporabnik lahko namesti z ukazom `pip install`. Za potrebe testiranja pa se je paket namestilo lokalno s pomočjo orodja `wheel`. Odjemalni program od uporabnika zahteva vrsto podatkov o začetni in končni točki prehoda in glede na podatke, vnesene v prvem koraku, prilagodi vnaprej zastavljena vprašanja. V vsakem koraku lahko uporabnik namesto podatka vnese črko „h“ kot (*ang.* help) in dobi dodatno razlago o parametru. Na ta način uporabniku olajšamo delo, saj je voden čez celoten proces.

Program za uporabnika postavi funkcijo na platformi začetne točke, ki je lahko na uporabniškem računu uporabnika ali na računu ponudnika. Funkcija nato prejme obvestilo od mikro storitve začetne točke in ga preusmeri na mikro storitev končne točke.

Ker dogodkovni prehod še ni ponujen kot storitev na enem uporabniškem računu ali računu podjetja, ima orodje naslednja ukaza, kjer uporabnika vpraša, na kakšen način želi gostiti funkcijo.

```
$ event-gateway-cli configure-aws
```

```
$ event-gateway-cli configure-azure
```

Funkcijo je mogoče gostiti na istem uporabniškem računu kot aplikacijo uporabnika, ali na drugem uporabniškem računu, ki je namenjen samo dogodkovnem prehodu. Uporabnik nato poda dostopne podatke do računa, ki bo gostil dogodkovni prehod, in dostopne podatke do računa aplikacije. Če uporabnik gosti svoj dogodkovni prehod, je potreben le vnos dostopnih podatkov do tega računa. Primer za nastavitev AWS kot ponudnika začetne točke:

```
$ event-gateway-cli configure-aws
How would you like to host the event gateway?
1. On a service account
2. On the user account
Please select the option you want (1 or 2): 1
Service account access key: xxxxxxxx
Service account secret key: yyyyyyyy
User account access key: xxxxxxxx
User account secret key: yyyyyyyy
```

V prihodnje bi ponudnik storitve lahko, zaradi varnosti, kodo orodja prestavil v oblak, kjer bi s pomočjo tehnologije API ponudil urejen dostop za registrirane uporabnike in tako zagotovil varno uporabo orodja.

4.5 Razširjena funkcionalnost

V tem poglavju bodo predstavljeni koraki, ki so potrebni za razširitev dogodkovnega prehoda. Za primer bomo vzeli novo mikro storitev in nastavili dogodkovni prehod, da bo podpiral dogodke te storitve. Osnovna rešitev je zasnovana tako, da je dodajanje nove mikro storitve trivialno.

4.5.1 Razširjen odjemalni program za konfiguracijo dogodkovnega prehoda

Za primer nove začetne točke pri AWS potrebujemo mikro storitev `API Gateway`, katero smo do zdaj naslavljali iz funkcije dogodkovnega prehoda. Tokrat `API Gateway` upoštevamo kot začetno točko prehoda oziroma kot mikro storitev, ki bo poslala dogodek. V odjemalnem programu dodamo `API Gateway` kot možno začetno točko in ob uporabnikovem vnosu `API Gateway` nastavimo novo metodo, katera je naslovljena na funkcijo dogodkovnega prehoda. Uporabnik mora vpisati podatke o `API gateway` in uporabniškem računu, kjer se nahaja začetna točka. Ko je metoda definirana, moramo funk-

ciji dogodkovnega prehoda dodati dovoljenje za dostopanje iz uporabniškega računa uporabnika in mikro storitve. Nato objavimo metodo, ki preveri, če ima dostop do ciljne točke. Če ga nima, objava spodleti. Prehod lahko testiramo tako, da s pomočjo programa `Postman`, ali podobnega programa, pokličemo metodo znotraj `API Gateway`. Vsebino dogodka lahko tokrat določimo v parametru `body` zahtevka, ki ga pošlje program Postman. Kot vidimo je rešitev zasnovana tako, da je razširljiva na novo mikro storitev v le nekaj korakih. To je zelo pomembno, saj je mikro storitev veliko in si pred objavo želimo podpreti vsaj osnovne mikro storitve. V vsakem primeru bomo naleteli na mikro storitve, za katere bo dodajanje podpore drugačno, vendar z razširljivo rešitvijo poskrbimo za večino. Pri ponudniku Amazon hitro vidimo, če mikro storitev spada med preprosto razširljive storitve tako, da preverimo, če mikro storitev omogoča preusmerjanje dogodka s pomočjo ARN (*ang.* amazon resource name).

Pri dodajanju novih začetnih točk na Microsoft Azure je potrebna le nova konfiguracija EventGrid poslušalca sporočil, saj so vsi dogodki znotraj Microsoft Azure lahko preusmerjeni na storitev EventGrid. To nam močno olajša dodajanje novih začetnih mikro storitev za dogodkovni prehod.

4.5.2 Razširjena funkcija dogodkovnega prehoda na AWS

Funkcijo dogodkovnega prehoda je potrebno razširiti tako, da ugotovi, od katere mikro storitve je prejela zahtevek, in glede na to razčleni podatke. Za primer smo vzeli API gateway, da vidimo izjemo, kjer dogodek ni generiran iz strani ponudnika, temveč ga poda uporabnik. V tem primeru moramo v dokumentaciji definirati, kako mora biti dogodek sestavljen v primeru, da ga mora uporabnik vnesti sam. Za osnovo vzamemo dogodek, ki ga zgenerira AWS in ga priredimo za primer API prehoda.

4.5.3 Razširjena funkcija dogodkovnega prehoda na Microsoft Azure

Pri razširitvi prehodne funkcije pri ponudniku Microsoft Azure ne rabimo definirati nove strukture dogodka, saj Microsoft Azure uporablja storitev EventGrid, kar pomeni, da vse dogodke, ki jih zgeniririajo mikro storitve, preusmeri na storitev EventGrid, ki priredi strukturo dogodka. Tako lahko iz JSON dogodka ugotovimo, katera mikro storitev je poslala sporočilo in druge pomembne informacije. V funkciji tako napišemo pogoje, ki prepoznajo začetno mikro storitev in glede na to informacijo razčlenijo podatke.

Poglavje 5

Rezultati razvoja

5.1 Rezultati

Rezultat razvoja in diplomskega dela je osnova sistema, ki je namenjen prenašanju dogodkov med težko povezljivimi mikro storitvami v oblaku. Končna rešitev je implementirana kot odjemalni program, ki uporabniku omogoča nastavitev dogodkovnega prehoda za lastne potrebe. Odjemalni program uporabnika vodi čez nastavitev dostopnih podatkov, nastavitev gostitelja dogodkovnega prehoda in nastavitev podatkov o začetni ter končni točki prehoda. Tako uporabniku močno skrajšamo čas, ki je potreben za postavitev takšne rešitve, in mu omogočimo, da upošteva mikro storitve pri drugih ponudnikih kot možnost pri razvoju svoje rešitve. Dogodkovni prehod od uporabnika zahteva le, da zna poiskati zahtevane podatke o vpletenih mikro storitvah na platformi ponudnika. Osnova rešitve je koristna za podjetja, ki potrebujejo dogodkovni prehod, a ne želijo razviti svojega (denarna ali časovna stiska). Za postavitev lahko uporabijo odjemalni program, vendar morajo dogodkovni prehod gostiti na svojem računu, saj še ni ponujen kot storitev. To podjetjem prihrani sredstva, saj je odjemalni program preprost za uporabo in ne zahteva nobenega predhodnega znanja. Rešitev se lahko uporablja za sestavo aplikacij, ki uporabljajo mikro storitve, katerim med sabo ni potrebno komunicirati. Uporabljene mikro storitve morajo biti

zmožne generiranja dogodkov in komunikacije s pomočjo dogodkov ter se morajo nahajati na platformi AWS ali Microsoft Azure. Kot dodatni primer uporabe lahko vzamemo aplikacijo, ki na eni oblačni platformi zbira dogodke iz IoT naprav in jih nato posreduje drugi platformi za izpis in monitoriranje. Podjetje se za tako rešitev odloči, saj je monitoriranje na drugi platformi bolj razvito ali bolj primerno za njihovo rešitev. Aplikacijo sestavimo tako, da začetno točko (IoT zaznavanje dogodkov iz naprav) povežemo na dogodkovni prehod in tako nanj preusmerimo dogodke. Dogodkovni prehod nato glede na zahteve uporabnika dogodke pošlje na končno točko, kjer se izpišejo informacije o zaznanih dogodkih. Tako združimo močni točki obeh ponudnikov s ciljem razvoja bolj kakovostne aplikacije.

5.2 Kako naprej

V nadaljevanju razvoja rešitve bi morali razmisliti o vrsti za prejete dogodke v funkciji dogodkovnega prehoda. Omejitev pri AWS je 8 zahtevkov na enkrat. Tako lahko v trenutni rešitvi pri veliki obremenitvi pride do kolizije dogodkov in se lahko zato določeni dogodki izgubijo. Microsoft Azure pa ponuja do 10 vzporednih izvajanj, vendar je za to funkcionalnost potrebno funkcijo pravilno nastaviti. V prihodnje bi lahko rešitev dogodkovnega prehoda razširili na več mikro storitev AWS in Microsoft Azure ponudnikov, ki komunicirajo z dogodki. To bi pomenilo dodatne konfiguracije dovoljenj za funkcijo dogodkovnega prehoda in prepoznavanje mikro storitve v kodi funkcije, da se lahko ta primerno odzove.

Razmisliti je potrebno tudi o odzivnosti prehoda ob uporabi s strani več uporabnikov. Vsakemu uporabniku se ob uporabi nastavi funkcija z ID uporabniškega računa uporabnika v imenu funkcije. S tem omogočimo, da vsak uporabnik prejme svojo unikatno funkcijo. Če uporabnik uporablja več prehodnih nastavitev, lahko uporablja isto funkcijo za prenos dogodka. Funkciji se bodo dodale pravice za dostop več mikro storitev. Funkcija se bo glede na podatke v dogodkovnem paketu odločila, na katero končno točko mora

poslati poizvedbo.

Pri oblačnih aplikacijah nam prav pride tudi avtomatizacija in orkestracija rešitve, zato bi se v prihodnje lahko razvilo ansible zbirko za dogodkovne prehode med ponudniki, ki bi omogočala definiranje dogodkovnega prehoda znotraj ansible skripta. To bi za uporabnika pomenilo, da lahko nastavi in naloži celotno aplikacijo znotraj ansible skripta in ob ponovni postavitvi aplikacije ne potrebuje ponovnega ročnega nastavljanja dogodkovnega prehoda s pomočjo odjemalnega programa.

5.3 Diskusija

Uporaba oblačnih ponudnikov ima precej strmo krivuljo učenja. Vsak ponudnik ima veliko število različnih mikro storitev in drugače zasnovan uporabniški vmesnik. Že učenje uporabe same platforme nam vzame veliko časa. Za veliko rešitev je dovolj uporaba le grafičnega vmesnika oziroma platforme ponudnika, za kompleksnejše rešitve ali konfiguracije pa potrebujemo še druga orodja. Pri ponudniku Microsoft Azure opazimo pomanjkljivo dokumentacijo za njihova orodja, saj so bolj osredotočeni na zagotavljanje storitev skozi uporabo uporabniških vmesnikov. Zaradi same kompleksnosti tematike diplomskega dela in želje po avtomatizaciji procesa, uporaba uporabniških vmesnikov ni bila mogoča.

Razvoj rešitev v oblaku je precej zahteven in počasen proces zaradi omejenih možnosti lokalnega testiranja in časovno potratnega postavljanja aplikacij. Zato je pomembno, da razvijalec poskrbi za dobro zasnovane sprotne izpise informacij v kodi, iz katerih je jasno, kje je prišlo do težave. Prav tako je zelo pomemben proces snovanja rešitve, saj, če nismo previdni, hitro naletimo na slepo ulico in moramo zavreči del razvite rešitve. To še posebej velja za oblak, saj v veliko primerih težko razvijemo alternativne korake do rešitve.

Poglavje 6

Zaključek

Eden izmed večjih problemov v današnji industriji, ne le na področju oblačnih storitev, je pomanjkanje standardnih protokolov, zato različni ponudniki sistem definirajo tako, da najbolje služi lastnih potrebam. Iz vidika ponudnika takšna rešitev zmanjša stroške in poenostavi implementacijo. Iz vidika končnega uporabnika in razvijalcev to predstavlja omejitev, saj je menjava posamezne mikro storitve ali celotnega ponudnika dolgotrajen in drag proces. Zato odvisnost od ponudnika predstavlja resen problem, saj so uporabniki zaradi omejenih možnosti prisiljeni v razvoj manj kakovostne rešitve.

Dogodkovni prehod med vodilnimi ponudniki oblaka je zato zelo zaželena storitev, vendar je ponudnikom oblaka v večjem interesu reševati težave uporabnikov na način, da jih obdržijo. Zaradi kompleksnosti razvoja dogodkovnega prehoda bi bilo smiselno najprej razviti dogodkovni prehod za manjše število mikro storitev in poskrbeti, da prehod deluje dobro. Kljub temu, da bi bila rešitev primerna za manjšo ciljno publiko, bi omogočala ravno to, kar nekateri potrebujejo.

V diplomskem delu se dotaknemo omenjenega področja, kjer s pomočjo dodatnih komponent povežemo različne mikro storitve, ki sicer niso povezljive, in tako omejimo posledice omenjenega problema. Področje je še zelo mlado in neraziskano, saj imajo dosedanji poskusi veliko težav in pomanjkljivosti. V delu definiramo osnovo sistema, ki s pomočjo modernih ogrodij

definira preprosto in hitro postavitev dogodkovnega prehoda za uporabnika. Prehod je strukturiran modularno, kar omogoči trivialno razširitev na nove mikro storitve.

Literatura

[1] Avoiding cloud vendor lockin. Dosegljivo: https://www.thorntech.com/2017/09/avoidingcloudvendorlockin/. [Dostopano: 2. 6. 2020].

[2] Azure event grid. Dosegljivo: https://docs.microsoft.com/en-us/azure/event-grid/overview. [Dostopano: 8. 6. 2020].

[3] Block file and object storage difference. Dosegljivo: https://www.caringo.com/blog/back-basics-object-storage, note = [Dostopano: 16. 8. 2020],.

[4] Cloud events. Dosegljivo: https://cloudevents.io/, note = [Dostopano: 22. 7. 2020],.

[5] Nicola Dragoni, Saverio Giallorenzo, Alberto Lluch Lafuente, Manuel Mazzara, Fabrizio Montesi, Ruslan Mustafin, and Larisa Safina. Microservices: yesterday, today, and tomorrow. In *Present and ulterior software engineering*, pages 195–216. Springer, 2017.

[6] Event computing. Dosegljivo: https://en.wikipedia.org/wiki/Event_(computing). [Dostopano: 23. 5. 2020].

[7] Event driven programming. Dosegljivo: https://en.wikipedia.org/wiki/Event-driven_programming. [Dostopano: 23. 5. 2020].

[8] History of aws. Dosegljivo: https://techcrunch.com/2016/07/02/andy-jassys-brief-history-of-the-genesis-of-aws/, note = [Dostopano: 1. 8. 2020],.

[9] Lambda. Dosegljivo: https://aws.amazon.com/lambda/. [Dostopano: 23. 5. 2020].

[10] Life before cloud computing. Dosegljivo: https://www.ukessays.com/essays/information-technology/life-before-cloud-computing-information-technology-essay.php. [Dostopano: 23. 5. 2020].

[11] Brenda M Michelson. Event-driven architecture overview. *Patricia Seybold Group*, 2(12):10–1571, 2006.

[12] Dmitry Namiot and Manfred Sneps-Sneppe. On micro-services architecture. *International Journal of Open Information Technologies*, 2(9):24–27, 2014.

[13] Object storage. Dosegljivo: https://www.infoworld.com/article/2614094/what-is-object-storage-.html. [Dostopano: 23. 5. 2020].

[14] Justice Opara-Martins, Reza Sahandi, and Feng Tian. Critical analysis of vendor lock-in and its impact on cloud computing migration: a business perspective. *Journal of Cloud Computing*, 5(1):4, 2016.

[15] Postman. Dosegljivo: https://www.postman.com/, note = [Dostopano: 18. 8. 2020],.

[16] Python package index. Dosegljivo: https://en.wikipedia.org/wiki/Python_Package_Index, note = [Dostopano: 5. 7. 2020],.

[17] State of the cloud report. Dosegljivo: https://resources.flexera.com/web/media/documents/rightscale-2019-state-of-the-cloud-report-from-flexera.pdf, note = [Dostopano: 16. 8. 2020],.

[18] Vendor lockin. Dosegljivo: https://www.cloudflare.com/learning/cloud/what-is-vendor-lock-in/. [Dostopano: 27. 5. 2020].

[19] What is ansible. Dosegljivo: https://opensource.com/resources/what-ansible. [Dostopano: 26. 6. 2020].

[20] xopera. Dosegljivo: https://github.com/xlab-si/xopera-opera, note = [Dostopano: 18. 8. 2020],.