Arhitektura funkcija kao servisa

FaaS, *Serverless*

1

Sadržaj

- Arhitektura funkcija kao servisa
- Primeri aplikacija (AWS)

2

Arhitektura funkcija kao servisa

3

Arhitektura funkcija kao servisa

- Motivacija
  - skupo korišćenje stalno pokrenutih *cloud* resursa za povremenu obradu podataka
    - obrada podataka kao reakcija na događaj
    - obrada podataka u određenom vremenskom trenutku
  - naporno ručno pokretanje i zaustavljanje servisa
    - u cilju uštede resursa
  - naporno konfigurisanje i održavanje ručno pokrenutih servisa
  - ručno pokrenute instance provode dosta vremena u pasivnom stanju
  - teška ili skupa integracija dinamičke orkestracije i pružaoca *cloud* usluga
    - npr. integracija Kubernetes-a u AWS-a
  - **rešenje**: dinamičko upravljanje funkcijama kao servisima
    - kolokvijalno engl. ***Serverless***

4

4

Arhitektura funkcija kao servisa

- *Serverless*
  - zbunjujući naziv jer indikuje nepostojanje serverskog hardvera i serverskih procesa
    - koji zapravo postoje
    - ali njima **ne upravlja organizacija koja razvija softver**
  - ne postoji jedinstvena definicija pojma, već se podrazumevaju dva aspekta:
    - engl. *Backend as a Service,* BaaS
      - istorijski prva upotrebljena definicija
      - upotreba postojećih servisa radi implementiranja serverske podrške za aplikacije
        - ne čuvaju stanje aplikacije
        - npr. Firebase i Auth0
    - engl. *Function as a Service*, FaaS
      - razvoj serverskih komponenti visokog nivoa granularnosti
        - ne čuvaju stanje aplikacije
        - pokreću se kao reakcija na odgovarajuće događaje

5

5

Arhitektura funkcija kao servisa

- *Serverless* - primer web prodavnice opreme za kućne ljubimce
  - potrebno da podržava autentifikaciju, pretragu kataloga (po obeležijima i po reči u tekstu) i kupovinu proizvoda
  - tradicionalni pristup razvoju aplikacija:

izvor: *Serverless Architectures, Mike Roberts, online:* https://martinfowler.com/articles/serverless.html

6

6

Arhitektura funkcija kao servisa

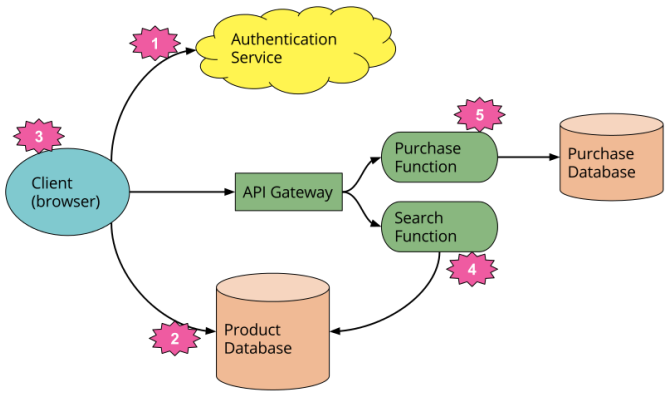
- *Serverless* - primer web prodavnice opreme za kućne ljubimce
  - *serverless* pristup razvoju aplikacija:
    - zasnovan na mikroservisnim arhitekturama
    - eliminacija autentifikacije i korišćenje eksternog servisa (npr. Auth0)
    - izdvajanje baze podatka o proizvodima u poseban BaaS (npr. Firebase)
    - pretraga je izdvojena u funkciju koja se izvršava po potrebi
      - na svaki HTTP zahtev za pretragom
    - kupovina je posebna funkcija koja se izvršava po potrebi
      - ostala na serverskoj strani iz sigurnosnih razloga
    - klijentska aplikacija postaje svestan različitih elemenata arhitekture
      - mora da održava sesiju korisnika
      - najčešće SPA (engl. *single page application*)

7

7

Arhitektura funkcija kao servisa

- *Serverless* - primer web prodavnice opreme za kućne ljubimce
  - *serverless* pristup razvoju aplikacija:



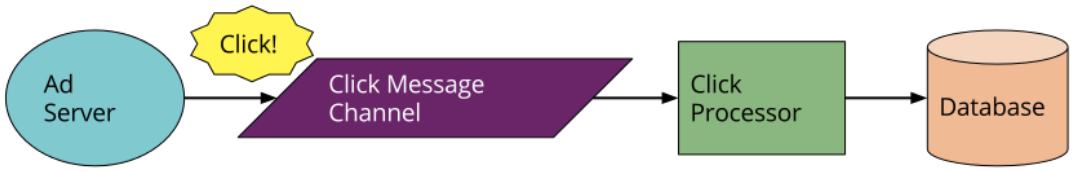
izvor: *Serverless Architectures, Mike Roberts, online:* https://martinfowler.com/articles/serverless.html

8

8

Arhitektura funkcija kao servisa

- *Serverless* - primer obrade korisničkih akcija na webu
  - npr. obrada korisničkog odabira reklame na stranici i redirekcija korisnika
  - tradicionalni pristup
    - redirekcija korisniku i asinhrono slanje poruke obrađivaču podataka



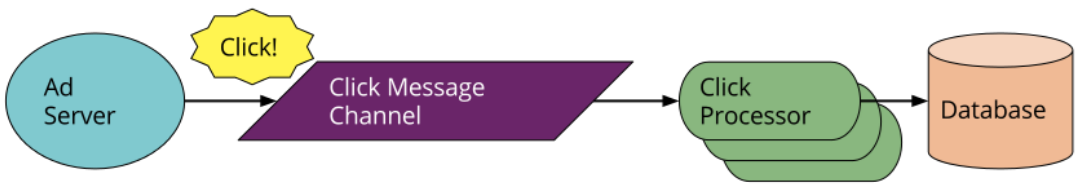
izvor: *Serverless Architectures, Mike Roberts, online: https://martinfowler.com/articles/serverless.html*

9

9

Arhitektura funkcija kao servisa

- *Serverless* - primer obrade korisničkih akcija na *web*-u
  - *serverless* pristup
    - pokretanje obrađivača po potrebi
    - paralelizacija obrade podataka
    - pružalac *cloud* usluge nudi i brokera za razmenu poruka i okruženje za dinamičko pokretanje i zaustavljanje funkcija programa



izvor: *Serverless Architectures, Mike Roberts, online: https://martinfowler.com/articles/serverless.html*

10

10

Arhitektura funkcija kao servisa

- *Serverless*, FaaS
  - pristup razvoju i upravljanju aplikacijama bez upravljanja serverima
  - pristup razvoju i upravljanju aplikacijama bez postojanja dugotrajnih aplikacija
    - ključna razlika u odnosu PaaS (engl. *Platform as a Service*)
  - nije potreban nikakav poseban radni okvir ili biblioteka
    - funkcije su samostalne aplikacije u podržanom programskom jeziku
    - obično postoje ograničenja u trajanju i količini resursa koji se mogu dobiti
  - za pokretanje funkcije je potreban samo aplikativni kod
    - bez programske podrške za upravljanje kontejnerima, VM-ovima ili hardverom
  - horizontalno skaliranje je ugrađeno u uslugu i transparentno
    - bez potrebe za eksplicitnim podešavanjem i upravljanjem
  - izvršavanje funkcija inicirano događajima
    - iako moguće ručno pokretanje funkcije, najčešće u fazama testiranja
    - npr. vreme, korisnički zahtev, događaj u toku podataka

11

11

Arhitektura funkcija kao servisa

- Upravljanje stanjem
  - ograničene mogućnosti čuvanja stanja u okviru infrastrukture za pokretanje ovakvih aplikacija
    - postoje mogućnosti pisanja u memoriji ili na disk
    - ne postoji garancija očuvanja vrednosti između dva pokretanja
  - po svojoj prirodi **funkcije ne čuvaju stanje**
    - potpuno funkcionalne transformacije podataka
    - ukoliko je potrebno, stanje čuvati izvan izvršnog okruženja funkcije
      - u skladištu koje podržava visoko konkurentni rad
    - npr. eksterne baze podataka, *cache* sistemi

12

12

Arhitektura funkcija kao servisa

- Trajanje izvršavanja
  - postoji unapred određeno maksimalno vreme izvršavanja funkcija
    - predefinisano od strane pružaoca *cloud* usluga
    - moguće podesiti maksimalno vreme izvršavanja do unapred definisanog maksimuma
    - trenutno ~15min

13

13

Arhitektura funkcija kao servisa

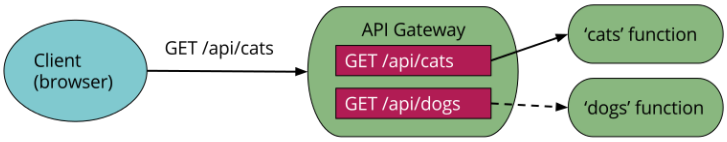
- Vreme pokretanja
  - potrebno vreme da se pokrene funkcija
    - engl. *startup latency*
    - od nekoliko milisekundi do nekoliko sekundi
  - **toplo pokretanje** (engl. *warm start*)
    - ponovno pokretanje prethodno kreiranih instanci radi uštede vremena
  - **hladno pokretanje** (engl. *cold start*)
    - kreiranje novih instanci funkcija
    - periodično se događa iako su funkcije prethodno pokretane
      - ukoliko je istekao period čuvanja instance u *cache*-u
    - trajanje zavisi od mnoštva faktora
      - programskog jezika, inicijalizacije biblioteka, eksternih servisa itd.

14

14

Arhitektura funkcija kao servisa

- *API gateway*
  - rutiranje korisničkih zahteva ka funkcijama koje ih obrađuju
  - rutiranje odgovora od funkcija ka korisnicima koji su poslali zahtev
- Alati za rad i povećanje produktivnosti
  - postoje radni okviri kao što je Serverless, Claudia, Zappa
    - koji apstrahuju *cloud* platformu
    - otvorenog koda
      - moguće plaćanje za napredne funkcionalnosti ili podršku



izvor: *Serverless Architectures, Mike Roberts, online: https://martinfowler.com/articles/serverless.html*

15

15

Arhitektura funkcija kao servisa

- FaaS i PaaS i kontejneri
  - PaaS nisu napravljene za pokretanje i zaustavljanje celih aplikacija kao odgovor na eksterni događaj
  - skaliranje aplikacije u PaaS i kontejnerizovanom okruženju zahteva podešavanje
    - kod PaaS potrebno je konfigurisati platformu da bi obezbedila odgovarajuće skaliranje aplikacije
      - automatsko skaliranje neće i dalje skalirati na nivou pojedinačnog zahteva
    - kod kontejnera potrebno održavati orkestrator
      - upravljanje FaaS platformom je u nadležnosti pružaoca *cloud* usluga
    - kod FaaS skaliranje je transparentno
      - i povoljnije sa stanovišta cene rada funkcije
  - **mala razlika između FaaS i kontejnera sa dinamičkom orkestracijom**
    - količina konfiguracije
    - više je poslovna odluka

16

16

Arhitektura funkcija kao servisa

- Prednosti
  - redukovana cena rada aplikacije
    - redukovana cena razvoja softvera (BaaS)
    - redukovana cena skaliranja
  - pojednostavljeno upravljanje životnim ciklusom aplikacije
  - "zelenije" korišćenje računarskih resursa

17

17

Arhitektura funkcija kao servisa

- Prednosti - redukovana cena rada aplikacije
  - rad aplikacije bez potrebe za upravljanjem infrastrukturom
    - serverima, bazama podataka, i delom aplikativne logike
    - dokle god pružalac *cloud* usluga poseduje date servise (BaaS)
      - manje se plaća jer se upravlja velikim brojem sličnih instanci na sličan način
  - smanjenje cene usled
    - korišćenja deljene infrastrukture umesto namenske
    - manji broj ljudi koji učestvuju u razvoju i eksploataciji aplikacije

18

18

Arhitektura funkcija kao servisa

- Prednosti - redukovana cena rada aplikacije
  - **redukovana cena razvoja softvera** (BaaS)
    - korišćenje BaaS pojednostavljuje i ubrzava razvoj softvera
      - cele aplikacije pružene kao servisi
      - za razliku od IaaS i Paas
        - gde su serveri u operativni sistemi pruženi kao servisi
      - npr. autentifikacija, baza podataka

19

19

Arhitektura funkcija kao servisa

- Prednosti - redukovana cena rada aplikacije
  - **redukovana cena skaliranja**
    - plaća se utrošena procesna moć u jedinici vremena
      - vrlo kratak obračunski period ~1ms
      - isplativost zavisi od količine i distribuiranosti saobraćaja
    - primer retkih zahteva
      - nema potrebe za stalno pokrenutim serverom
      - verovatnoća *free tier* obrade
    - primer neuniformnog saobraćaja
      - nema potrebe za zakupljivanjem hardvera koji može da obradi zahteve kada je broj na maksimumu

izvor: *Serverless Architectures, Mike Roberts, online:* https://martinfowler.com/articles/serverless.html

20

20

Arhitektura funkcija kao servisa

- Prednosti - pojednostavljeno upravljanje životnim ciklusom aplikacije
  - automatsko skaliranje resursa
    - na nivou pojedinačnog upita/korisničkog zahteva
  - smanjena kompleksnost distribucije kôda aplikacije
    - samo se izvršiv kôd aplikacije zapakuje u arhivu ili kontejner
    - arhiva ili kontejner se kopiraju na unapred definisano mesto na platformi
      - distribucija može biti automatizovana
  - posledica
    - kraće vreme potrebno da proizvod dođe na tržište
    - velika mogućnost i brzina eksperimentisanja sa novim funkcionalnostima

21

21

Arhitektura funkcija kao servisa

- Prednosti - “zelenije” korišćenje računarskih resursa
  - bolje i efikasnije iskorišćenje serverskih centara
    - manje potrošene energije
    - manje proizvedene toplote
  - bolji algoritmi za upravljanje hardverom
    - isključivanje dela hardvera do perioda dana sa najvećim opterećenjem

22

22

Arhitektura funkcija kao servisa

- Mane
  - mane u vezi sa platformom
    - moguće veliko kašnjenje između zahteva i odgovora
    - vezivanje za pružaoca *cloud* usluga
    - problemi sa paralelnim grupama korisnika
    - sigurnosni rizici
    - nemogućnost optimizacije serverske strane
    - nemogućnost čuvanja stanja između pokretanja FaaS
  - mane u vezi sa implementacijom

23

23

Arhitektura funkcija kao servisa

- Mane - moguće veliko kašnjenje između zahteva i odgovora
  - usled vremena potrebnog da se funkcija pokrene
    - hladan i topli start
  - usled broja servisa potrebnog da bi se korisnički zahtev rutirao
    - npr. API *Gateway*, IR *router*, FaaS

24

24

Arhitektura funkcija kao servisa

- Mane - vezivanje za pružaoca *cloud* usluga
  - kontrola nad serverima je u potpunosti u rukama vlasnika platforme
    - moguća promena cene, dostupnih konfiguracionih parametara
    - moguć pad infrastrukture i nedostupnost aplikacije
    - neizbežno unapređenje verzija biblioteka
  - zaključavanje za jednog pružaoca usluga
    - engl. *vendor lock-in*
    - migriranje koda na drugu platformu je možda nemoguće
      - zbog korišćenja specifičnih BaaS
      - zbog drugačijeg API-ja izvršnog okruženja
      - zbog drugačijih ograničenja koja postoje

25

25

Arhitektura funkcija kao servisa

- Mane - problemi sa paralelnim grupama korisnika
  - engl. *multitenancy problems*
  - instance softvera se izvršavaju na istom hardveru ili u istoj aplikaciji za različite grupe korisnika
    - potencijalni sigurnosni propusti
    - zagušenje sistema u nebalansiranom okruženju
    - iako retko, moguće je da se dogodi neki problem
- Mane - sigurnosni rizici
  - korišćenje svakog novog servisa povećava rizik od napada
  - direktno korišćenje BaaS **uklanja jedan podesiv sigurnosni sloj**
    - lakše je dospeti u aplikaciju i napraviti štetu
  - za svaku FaaS se mora podesiti sigurnosni profil
    - koji sadrži prava pristupa i potrebne dozvole nad resursima
    - što je veći broj funkcija veća je i verovatnoća pogrešnog podešavanja

26

26

Arhitektura funkcija kao servisa

- Mane - nemogućnost optimizacije serverske strane
  - direktno korišćenje BaaS uklanja srednji sloj
    - koji postoji između klijentske aplikacije i servisa
    - a koji često sadrži optimizovan kôd
  - kod FaaS je manje očito i veće su mogućnosti optimizacije
    - moguće je i deo funkcionalnosti izvući na pravi server
- Mane - nemogućnost čuvanja stanja između pokretanja FaaS
  - za neke slučajeve korišćenja esencijalno
  - oslanjanje na eksterne mehanizme za čuvanje stanja
    - moraju podržavati konkurentno pisanje i čitanje

27

27

Arhitektura funkcija kao servisa

- Mane - mane u vezi sa implementacijom
  - nemogućnost detaljne konfiguracije
  - nemogućnost posedovanja međustanja na serveru na kojem se funkcije izvršavaju
  - DoS sopstvene aplikacije
    - usled ograničenja platforme moguće je pokrenuti toliko funkcija da je nemoguće pokrenuti nove
  - nedovoljno trajanje izvršavanja i dugo pokretanje funkcija
  - teško integraciono testiranje
    - jedinično testiranje nije veliki problem
  - teško *debug*-ovanje
    - kod većine pružaoca *cloud* usluga
    - često nije podržano u produkcionom okruženju
      - oslanja se na logove

28

28

Arhitektura funkcija kao servisa

- Rešavanje mana
  - kreiranje boljih alata
  - upravljanje stanjem unutar funkcija
  - **unapređenje platformi za izvršavanje FaaS**
  - **definisanje šablona za kreiranje aplikacija**
  - **unapređenje procesa integracionog testiranja**

29

29

Arhitektura funkcija kao servisa

- Orkestracija funkcija
  - moguće je uvezivanje funkcija u koherentne celine
    - kreiranje aplikacija isključivo sastavljenih od nezavisnih funkcija
  - kreiranje sekvenci funkcija
    - gde je izlaz iz jedne funkcije ulaz u narednu
  - ponovno izvršavanje funkcija koje nisu dobro izvršene
  - uslovno izvršavanje funkcija
  - paralelno izvršavanje funkcija

30

30

Arhitektura funkcija kao servisa

- Pružaoci usluga FaaS
  - **AWS Lambda**
  - **Google Cloud Functions**
  - **Azure Functions**
  - **IBM OpenWhisk**
  - Alibaba Function Compute
  - Iron Functions
  - Auth0 Webtask
  - Oracle Fn Project
  - Kubeless
  - OpenFaaS
  - Zeit

31

Primeri aplikacija (AWS)

32

Primeri aplikacija (AWS)

- Obrada tokova podataka
  - bez *serverless* pristupa
    - Zookeeper + (Kafka -> Storm/Spark)
    - zasnovani na klasterima računara ili *cloud* platformama
      - potrebno voditi računa o skaliranju aplikacije
        - posebno u prisustvu neuniformnog opterećenja i saobraćaja
    - zasnovane na kontejnerima i dinamičkom orkestriranju
  - sa *serverless* pristupom
    - skup funkcija koje se automatski skaliraju
    - koriste se platformske opcije za prihvat i distribuciju podataka
      - npr. AWS Kinesis, AWS IoT, AWS SQS

33

33

Primeri aplikacija (AWS)

- Obrada tokova podataka - primeri šablona obrade
  - prikupljanje podataka sa senzora i jednostavna transformacija
    - IoT uređaji šalju merenja u komponentu za obuhvat podataka
    - merenja se transformišu u oblik pogodan za dalju obradu
    - primer upotrebe:
      - medicinska oprema šalje podatke o pacijentima
        - podaci moraju biti anonimizirani
  - obuhvat, transformacija i učitavanje podataka iz toka podataka
    - engl. *ingest, transform, load* (ITL)
    - dodavanje informacija svakom podatku
      - učitavanjem iz baze podataka ili statičkog skupa podataka
    - primeri upotrebe:
      - svakom podatku koji dođe sa senzora iz fabrike se dodaju podaci o senzoru
        - pročitani iz baze podataka ili GIS sistema
      - obrada zapisa u logovima i dodavanje konteksta

34

34

Primeri aplikacija (AWS)

- Obrada tokova podataka - primeri šablona obrade
  - analiza u realnom vremenu
    - u odnosu na prethodne šablone dodaje
      - agregacije nad vremenskim intervalom
      - detekciju anomalija nad tokom podataka
    - primeri upotrebe:
      - praćenje aktivnosti korisnika
      - analitika nad logovima
      - detekcija prevare
      - sistemi za davanje preporuka

35

35

Primeri aplikacija (AWS)

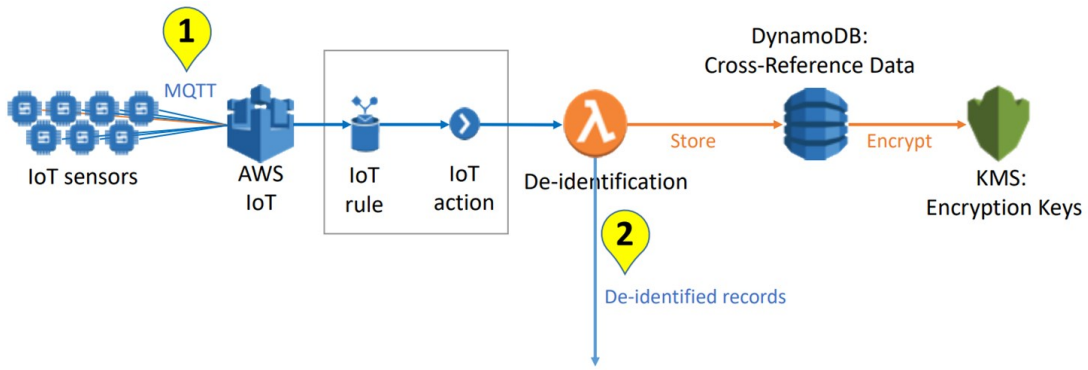
- Prikupljanje podataka sa senzora i jednostavna transformacija
  - medicinski uređaji (nepokretni uređaji, narukvice itd.) šalju podatke u realnom vremenu
    - potrebno obuhvatiti sve podatke
  - nakon prijema podataka potrebno anonimizirati podatke
    - zbog analize od strane drugih sistema
    - potrebno je svaki podatak opremiti ID-jem pacijenta
      - radi unakrsne analize podataka i praćenja stanja

36

36

Primeri aplikacija (AWS)

- Prikupljanje podataka sa senzora i jednostavna transformacija



izvor: *Serverless Streaming Architectures and Best Practices, online:* https://d1.awsstatic.com/serverless/Whitepaper/Stream%20Processing%20Whitepaper.pdf

Primeri aplikacija (AWS)

- Prikupljanje podataka sa senzora i jednostavna transformacija
  - medicinski uređaji šalju podatke po MQTT protokolu
    - IoT *Gateway* na nivou bolnice
    - podaci se zatim šalju AWS IoT servisu
      - na siguran način
  - IoT *rule*
    - prikuplja podatke od nekoliko pacijenata
      - mikro paket za anonimizaciju
  - IoT *action*
    - prosleđuje podatke novoj instanci lambde na anonimizaciju

Primeri aplikacija (AWS)

- Prikupljanje podataka sa senzora i jednostavna transformacija
  - AWS Lambda
    - uklanja identifikaciona polja iz podataka
    - smešta identifikaciona polja u Dynamo DB i dobija ID pacijenta
      - skladišteni podaci su kriptovani
      - potrebno radi upoređivanja rezultata među korisnicima
    - šalje podatke dalje u AWS Kinesis radi distribucije sistemima za analizu podataka

```
{
 "timestamp": "1508039751778",
 "device_id": "device8401",
 "patient_id": "patient2605",
 "name": "Eugenia Gottlieb",
 "dob": "08/27/1977",
 "temperature": 100.3,
 "pulse": 108.6,
 "oxygen_percent": 48.4,
 "systolic": 110.2,
 "diastolic": 75.6
}
```

```
{
 "timestamp": "1508039751778",
 "device_id": "device8401",
 "patient_id": "patient2605",
 "temperature": 100.3,
 "pulse": 108.6,
 "oxygen_percent": 48.4,
 "systolic": 110.2,
 "diastolic": 75.6,
}
```

39

39

Primeri aplikacija (AWS)

- Prikupljanje podataka sa senzora i jednostavna transformacija
  - razmatranja
    - kako bi se što više skratio topli start funkcija
      - koristiti statičke inicijalizacione blokove, *singleton* klase i globalne varijable
      - sve eksterne biblioteke zapakovati zajedno sa kodom
    - koristiti kompresiju podataka na izlazu iz lambda funkcije
    - podesiti veće *buffer*-e Kinesis servisa kako bi se povećala propusnost anonimiziranih podataka

40

40

Primeri aplikacija (AWS)

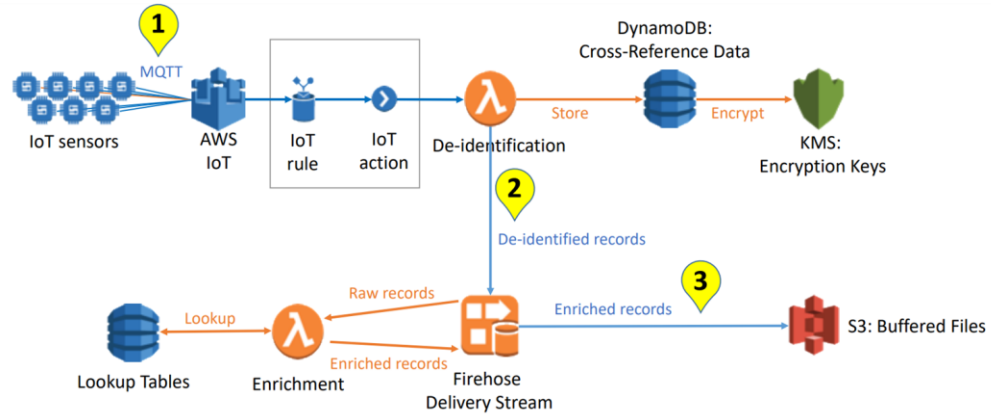
- Obuhvat, transformacija i učitavanje podataka iz toka (ITL)
  - nakon obuhvata podataka potrebno je
    - obogatiti podatke dodatnim poljima i vrednostima
    - zameniti neke enumeracije za potpune vrednosti

41

41

Primeri aplikacija (AWS)

- Obuhvat, transformacija i učitavanje podataka iz toka (ITL)



izvor: *Serverless Streaming Architectures and Best Practices, online:* https://d1.awsstatic.com/serverless/Whitepaper/Stream%20Processing%20Whitepaper.pdf

42

42

Primeri aplikacija (AWS)

- Obuhvat, transformacija i učitavanje podataka iz toka (ITL)
  - uvedena je nova AWS Lambda funkcija
    - inicirana od strane Kinesis servisa kada dođu novi slogovi
    - dodaje podatke o svakom mernom uređaju
    - formatira datumsko polje
    - šalje podatke nazad Kinesis servisu koji ih smešta u *buffer*
      - i šalje podatke dalje određenim servisima
      - šalje podatke na skladištenje i *backup* u S3
  - AWS Lambda skalira sa brojem podataka koji se nađu u Kinesis servisu
  - razmatranja
    - koristiti *cache* kako bi se ubrzalo traženje podataka o uređajima
      - Amazon DynamoDB Accelerator (DAX)

43

43

Primeri aplikacija (AWS)

- Obuhvat, transformacija i učitavanje podataka iz toka (ITL)

```
{
 "timestamp": "1508039751778",
 "device_id": "device8401",
 "patient_id": "patient2605",
 "temperature": 100.3,
 "pulse": 108.6,
 "oxygen_percent": 48.4,
 "systolic": 110.2,
 "diastolic": 75.6,
}
```

```
{
 "timestamp": "2018-01-27T05:11:50",
 "device_id": "device8401",
 "patient_id": "patient2605",
 "temperature": 100.3,
 "pulse": 108.6,
 "oxygen_percent": 48.4,
 "systolic": 110.2,
 "diastolic": 75.6,
 "manufacturer": "Manufacturer 09",
 "model": "Model 02"
}
```

44

44

44

Primeri aplikacija (AWS)

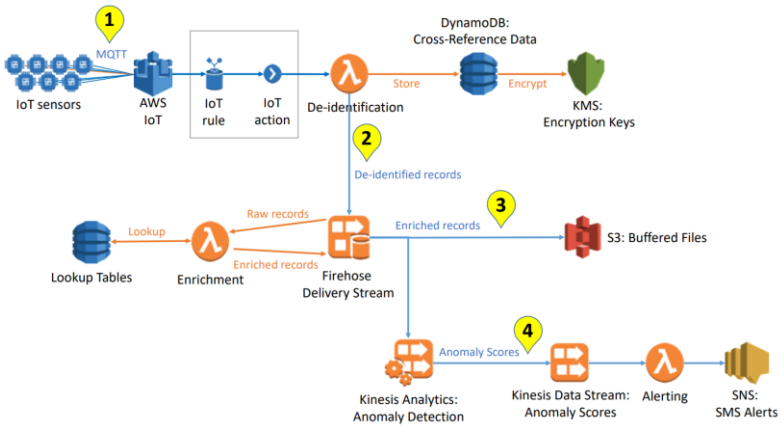
- Analiza u realnom vremenu
  - analiza podataka koje uređaji šalju radi detekcije anomalija u radu
    - ukoliko je anomalija detektovana šalje se notifikacija proizvođaču opreme
    - nadgledanjem šablona u porukama, proizvođači su u mogućnosti da predvide kvar
  - Kinesis Data Analytics
    - koristi se za detekciju šablona
    - rezultuje torkama označenim sa indikatorom anomalije
  - Kinesis Data Stream
    - predstavlja tok podataka koji je ulaz za novu Lambda funkciju
  - uvedena je nova AWS Lambda funkcija
    - na osnovu predefinisanih pragova tolerancije za anomaliju
      - šalje notifikaciju proizvođaču opreme o opremi koja je pred otkazom ili je u otkazu

45

45

Primeri aplikacija (AWS)

- Analiza u realnom vremenu



izvor: *Serverless Streaming Architectures and Best Practices, online:* https://d1.awsstatic.com/serverless/Whitepaper/Stream%20Processing%20Whitepaper.pdf

46

46

Primeri aplikacija (AWS)

- Analiza u realnom vremenu


```
{
 "timestamp": "2018-01-27T05:11:50",
 "device_id": "device8401",
 "patient_id": "patient2605",
 "temperature": 100.3,
 "pulse": 108.6,
 "oxygen_percent": 48.4,
 "systolic": 110.2,
 "diastolic": 75.6,
 "manufacturer": "Manufacturer 09",
 "model": "Model 02"
}
```

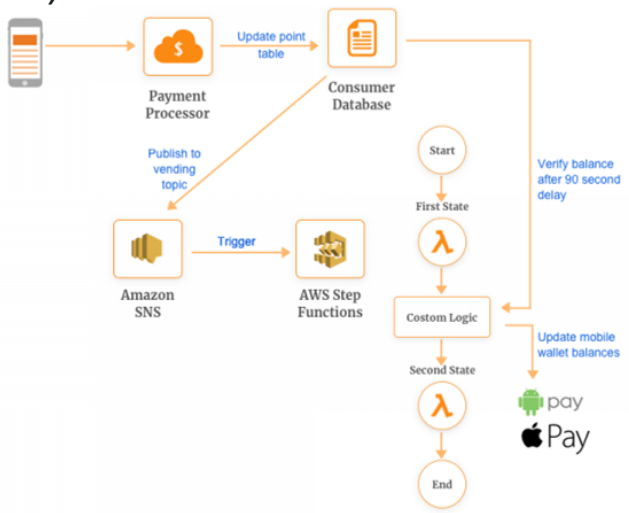


```
{
 "timestamp": "2018-01-27T05:11:50",
 "device_id": "device8401",
 "patient_id": "patient2605",
 "temperature": 100.3,
 "pulse": 108.6,
 "oxygen_percent": 48.4,
 "systolic": 110.2,
 "diastolic": 75.6,
 "manufacturer": "Manufacturer 09",
 "model": "Model 02",
 "anomaly_score": 0.9845
}
```


47

Primeri aplikacija (AWS)

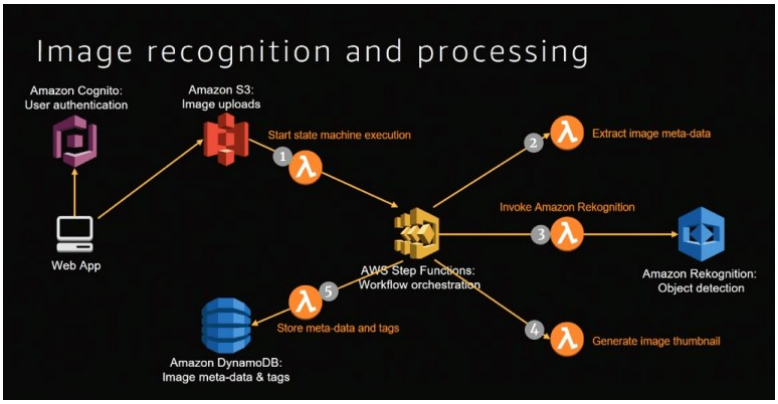
- Obrada plaćanja
  - implementacija tokova obrade podataka pomoću funkcija Lambda



izvor: *10 Practical Examples of AWS Lambda, online: https://www.simform.com/serverless-aws-lambda-examples/*

Primeri aplikacija (AWS)

- Prepoznavanje na slikama
  - 1. započinje proces obrade slike
  - 2. izvlači meta-podatke iz slike
  - 3. pokreće servis za prepoznavanje objekata u slikama
  - 4. kreira *thumbnail* slike
  - 5. zapisuje sve podatke u DynamoDB



izvor: *10 Practical Examples of AWS Lambda, online: https://www.simform.com/serverless-aws-lambda-examples/*

49

49

Literatura

- *Serverless Applications with Node.js - Using AWS Lambda and Claudia.js*, Slobodan Stojanović and Aleksandar Simović, Manning, 2019.
- *Serverless Architectures*, Mike Roberts, online: https://martinfowler.com/articles/serverless.html
- *Serverless Streaming Architectures and Best Practices*, online: https://d1.awsstatic.com/serverless/Whitepaper/Stream%20Processing%20Whitepaper.pdf

50

50