UNIVERZITET U NIŠU

ELEKTRONSKI FAKULTET

KATEDRA ZA RAČUNARSTVO

**Sistemi za upravljanje bazama podataka – Seminarski rad**

# Azure Cosmos DB kao primer DBaaS rešenja

Profesor:

Doc. dr Aleksandar Stanimirović

Student:

Nemanja Raković, 590

# Sadržaj

# 1. Uvod

U današnje doba je prepoznat značaj podataka kao potencijalno najvrednijeg resursa neke organizacije. S obzirom da se količina informacija koja je dostupna eksponencijalno uvećava, potrebno je imati alate koji mogu da na efikasni i korisniku efikasni način upravljaju masivnim, kompleksnim skupovima podataka i iz njih izvuku korisne informacije. U suprotnom, ti isti podaci mogu postati smetnja, s obzirom da cena njihovog skladištenja i održavanja može nadmašiti vrednost koju pružaju.

Gotovo sve organizacije, od lokalnih prodavnica do multinacionalnih korporacija i vlada, poseduju informacioni sistem, u čijoj srži se nalaze baze podataka, i sistemi za njihovo upravljanje. Baze podataka predstavljaju kolekciju informacija vezanih za određeni subjekat, namenu ili pojavu unutar jedne ili više povezanih organizacija. Sistem za upravljanje bazama podataka (*DBMS*) je softver namenjen da pomogne u održavanju i korišćenju velikih količina podataka.

DBMS ima nekoliko ključnih prednosti u odnosu na naivni pristup čuvanja podataka u datotekama i pisanja specifičnog aplikacionog koda za njihovo korišćenje. DBMS garantuje apstraktni pogled na podatke koji je nezavisan od stvarnog mehanizma skladištenja, sofisticirane algoritme i strukture za efikasno skladištenje i pristup podataka, integritet smeštenih podataka, kontrolu pristupa, korisničke interfejse za administraciju, konkurentni pristup bazi, konzistentni oporavak od pada sistema i funkcije koje su zajedničke velikom broju aplikacija koje koriste bazu te time omogućava brži razvoj klijentskog softvera.

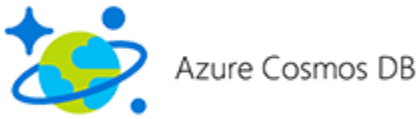
Da bi se postigla mala latencija i velika dostupnost podataka, instance aplikacija moraju da budu smeštene u data centre koji su blizu njihovim korisnicima. Zahtevi su uglavnom takvi da aplikacije treba da reaguju u realnom vremenu, da čuvaju velike količine podataka i da ih učine brzo dostupnim korisnicima

Tehnologije računarstva u oblaku (*cloud computing*) omogućuje korisnicima upotrebu *cloud* sistema baza podataka bez potrebe da kupuju i podešavaju sopstveni hardver, instaliraju DBMS softver i sami upravljaju samom bazom. Ovakvi servisi uglavnom vode računa o svim aspektima održavanja, od verzionisanja softvera do pružanja neprekidne dostupnosti servisa. Centri podataka su dostupni u više regiona čime se rešava problem latencije.

U nastavku rada će kao primer *cloud* baze podataka biti opisano Azure Cosmos DB DBaaS rešenje.

# 2. Azure Cosmos DB

Baza podataka kao servis (*Database as a Service,* u daljem tekstu *DBaaS*) se definiše kao paradigma namenjena upravljanju podataka u okviru koje spoljni (*third-party*) pružalac usluga obezbeđuje bazu podataka i softversku i hardversku podršku koja uz nju ide. [3]




*Slika 1 – Logo*

Azure Cosmos DB je potpuno upravljana (*fully-managed*) NoSQL baza podataka, koja služi za razvoj modernih aplikacija. Bazira se na *PaaS* (*Platform as a Service*) paradigmi.

Vreme odziva reda veličine milisekundi i automatska i instant skalabilnost garantuju brzinu na bilo kom nivou organizacije. Dostupnost je garantovana na veoma visokom nivou što osigurava kontinuitet poslovanja.

Razvoj aplikacija je brži i produktivniji zahvaljujući potpuno spremnom multi-regionskom distribuiranju podataka u celom svetu i API–ju otvorenog koda i SDK-ovima za popularne programske jezike.

Kao potpuno upravljani servis, Azure Cosmos DB preuzima odgovornost za administraciju baze podataka i održavanja same platforme, kao i upravljanje kapacitetom skladišta i na taj način omogućava aplikacijama potražnju resursa u zavisnosti od potreba kao i *serverless* pristup.

Azure Cosmos DB automatski upravlja podacima i njihovim porastom, tako što primenjuje horizontalnu particiju opterećenja. Ovaj proces je u potpunosti transparentan i pruža elastičnost i skaliranje virtuelno neograničenog skladišta, kao i dobar protok i brzinu.

Kao i drugi servisi, Azure Cosmos DB je distribuiran u više Microsoft data centara u svetu. U okviru jednog data centra, Azure Cosmos DB vrši automatsku replikaciju podataka, što omogućava visoku dostupnost podataka u tom regionu. Moguća je i replikacija podataka između data centara, odnosno globalno distribuiranje podataka – čime se postiže da se podaci nalaze bliže korisnicima.

# 3. Tipovi skladištenja

Azure Cosmos DB nudi nekoliko različitih interfejsa (API) za rad sa bazama podataka, kojima se podaci mogu modelovati podaci pomoću dokumenata, parova ključ-vrednost, grafova i *column-family* modela podataka. Na ovaj način se stiče utisak da Azure Cosmos DB predstavlja neku od navedenih tehnologija, bez dodatnih troškova upravljanja i skaliranja. Uz ove API-je je omogućeno korišćenje ekosistema, alata i postojećih veština za modeliranje podataka i kreiranje upita iz drugih sistema. Svi ovi pristupe nude automatsko skaliranje prostora za skladištenje i protoka, fleksibilnost i izuzetno dobre performanse

U ponudi su sledeće opcije:

- NoSQL API
- MongoDB API
- PostgreSQL API
- Cassandra API
- Gremlin API
- Table API

NoSQL API je razvijen za potrebe Azure Cosmos DB sistema i u najopštijem slučaju je preporučen pristup za rad sa platformom.

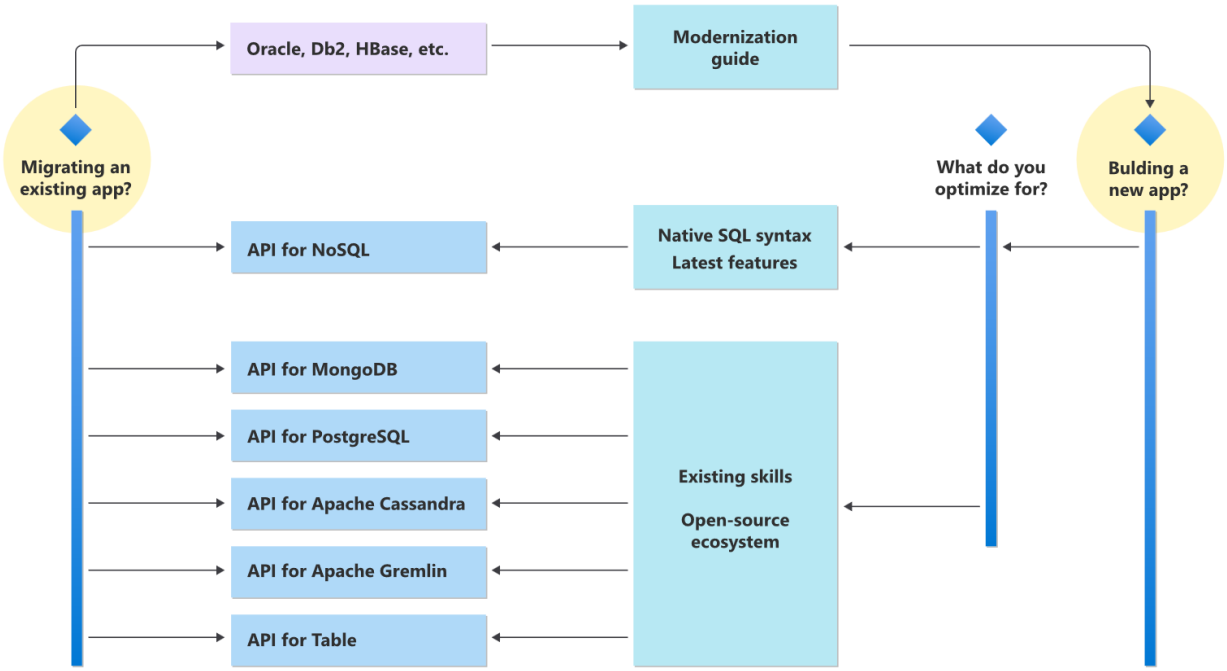
U slučaju da postoje aplikacije koje se oslanjaju na neke od drugih ponuđenih tehnologija za koje bi bilo potrebno prepisati ceo sloj za pristup podacima.

Takođe je moguće dalje korišćenje *open-source* ekosistema, resursa i znanja zbog kompatibilnosti protokola koji ovi API-ji koriste, dok su ključne funkcionalnosti Azure Cosmos DB platforme popu globalne distribucije podataka, performansi, niske latencije, skaliranja prostora i protoka i dalje dostupne.

Još jedan od razloga da se koristi neki drugi API osim NoSQL je za slučaj da aplikacije koriste sisteme računarstva u oblaku i drugih snabdevača osim Microsoft Azure-a, gde NoSQL API neće biti dostupan.

Pored kreiranja novih aplikacija, moguće je migrirati i podatke postojećih aplikacija, za šta je najčešće dovoljno samo izmeniti podešavanja za povezivanje na bazu podataka (*connection string*), pod uslovom da je prethodno proverena podrška API-ja za sve neophodne funkcionalnosti od kojih aplikacija zavisi.

Microsoft nudi sledeće preporučeno stablo odlučivanja prilikom izbora API-ja kod kreiranja novih ili migriranja postojećih aplikacija na Azure Cosmos DB:




*Slika 2 - Preporučeni proces odlučivanja za izbor Azure Cosmos DB API-ja*

## 3.1 NoSQL API

NoSQL API skladišti podatke u formi dokumenata i nudi najbolje krajnje korisničko iskustvo budući da postoji potpuna kontrola nad interfejsom, uslugom i SDK klijentskim bibliotekama. Svaka nova funkcionalnost koja se uvodi je prvo dostupna na nalozima za NoSQL API.

Ovaj API ima podršku za kreiranje upita korišćenjem SQL jezika za upite nad JSON objektima, dostupni su alati za analitiku i nudi izolaciju (u pogledu performansi) između operativnog i analitičkog opterećenja.

Ukoliko se vrši migracija sa baza poput Oracle, Db2, HBase ili DynamoDB, preporučeno je iskoristiti ovaj API.

## 3.2 MongoDB API

MongoDB API takođe skladišti podatke u formi dokumenata, sa tim što se koristi BSON format podataka i kompatibilan je sa MongoDB *wire* protokolom iako ne koristi izvorni kod MongoDB sistema i propratnih alata.

Ovaj API je odličan izbor ukoliko ima načina da se iskoristi znanja i veština vezanih za širi MongoDB ekosistem bez gubitka skaliranja, visoke dostupnosti, geo-replikacije, višestrukih lokacija za upis, automatskog transparentnog upravljanja *shardingom* i transparentnom replikacijom i drugih bitnih mogućnosti koje nudi Azure Cosmos DB.

Postojeće aplikacije mogu se iskoristiti samo promenom podešavanja za povezivanje na bazu podataka (*connection string*), a svi postojeći podaci se mogu premestiti pomoću MongoDB alata kao što su *mongodump* i *mongorestore* ili pomoću Azure Database Migration alata.

Standardni alati za rad sa MongoDB sistemom kao što su MongoDB shell, MongoDB Compass i Robo3T mogu i dalje da se koriste nakon prelaska na ovaj API.

Preporučena verzija MongoDB servera je 4.2, a minimalna dostupna 3.2.

## 3.3 Cassandra API

Cassandra API skladišti podatke u *column-oriented* šemi podataka koristeći protokol koji je kompatibilan sa Apache Cassandra rešenjem. Cassandra se zasniva na distribuiranom pristupu skladištenju velike količine podataka uz horizontalno skaliranje, uzimajući u obzir fleksibilnost *column-oriented* šema.

Ovaj API je najbolje izabrati ukoliko ima kada postoje mogućnosti da se iskoristi elastičnost i potpuno upravljanje koje Azure Cosmos DB nudi, ali i sve one izvorne funkcionalnosti koje Apache Cassandra nudi – distribuiranje, horizontalno skaliranje, fleksibilnost. Što znači da uz Cassandra API ne postoji potreba za upravljanjem operativnim sistemom, Java virtuelnom mašinom, Garbage kolektorom, klasterima, čvorovima i slično.

Omogućeno je korišćenje CQL-a (*Cassandra Query Language*), alata poput CQL Shell, ali i jedinstvenih funkcionalnosti koje Azure Cosmos DB pruža. Ovaj API trenutno podržava samo OLTP slučajeve.

## 3.4 Gremlin API

Gremlin API omogućava kreiranje graf upita i skladištenje podataka u vidu čvorova i potega. Koristi se u slučajevima koji uključuju dinamičke podatke sa složenim vezama, koje je nemoguće modelovati relacionom bazom, kao i kod aplikacija koje se već oslanjaju na Gremlin ekosistem.

Ovaj API kombinuje moć algoritama graf baza podataka sa visoko skalabilnom, potpuno upravljanom infrastrukturom, što daje odgovor na veliki broj problem vezanih za manjak fleksibilnosti relacionog pristupa.

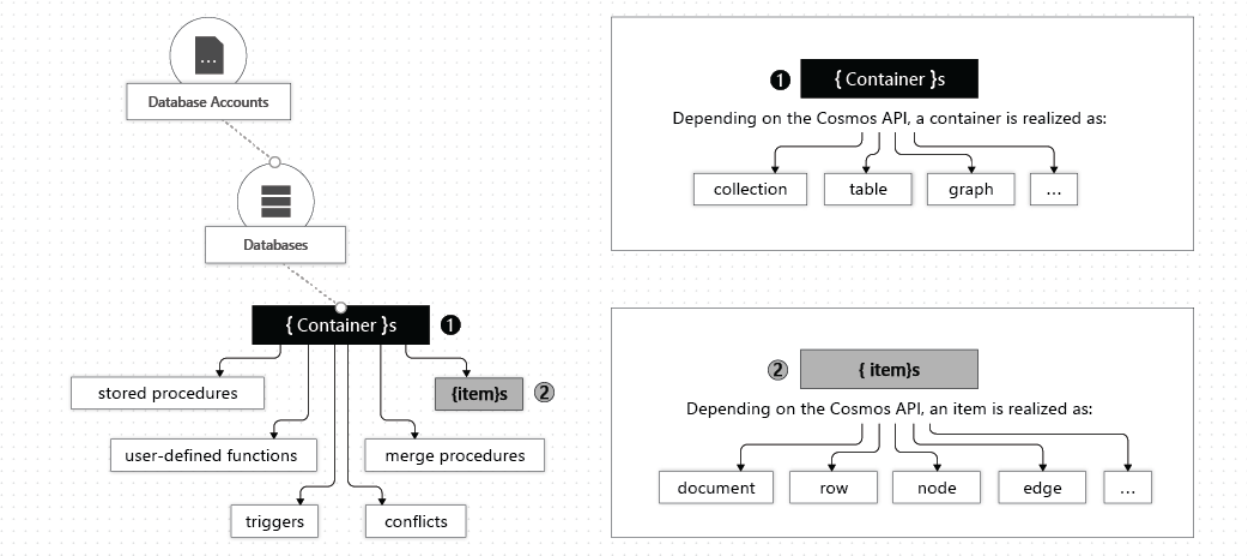
Gremiln API se oslanja na Apache TinkerPop rešenje za operacije nad grafovima i koristi isti GQL (*Graph Query Language*) za upis i izvršavanje upita. Podržan je Gremlin *wire* protokol i moguće je korišćenje Gremlink SDK-a, a za kompleksnu analizu grafova se mogu koristiti i ApacheSpark i GraphFrames tehnologije. Gremlin API trenutno podržava samo OLTP slučajeve.

## 3.5 Table API

Table API skladišti podatke u formi parova ključ-vrednost (*key-value pairs*). Kod Azure Table baze podataka postoje ograničenja u pogledu latencije, skaliranja, protoka, globalnog distribuiranja podataka i upravljanja indeksom. Međutim, Table API prevazilazi ova ograničenja i preporuka je migrirati na Azure Cosmos DB kako bi se iskoristile ove funkcionalnosti. U tom slučaju, aplikacije koje su pisane za Azure Table baze podataka, zahtevaju male promene koda. Table API takođe podržava samo OLTP slučajeve.

# 4. Upravljanje resursima

Azure Cosmos DB model resursa je hijerarhijski struktuiran. Osnovni element je Azure Cosmos korisnički nalog, koji obuhvata baze podataka, koje se sastoje od kontejnera. Svaki od ovih elemenata ima posebnu funkcionalnu ulogu. Na narednom dijagramu se može uočiti hijerarhija entiteta, kao i to kako određeni API-ji gledaju na apstrakcije kao što su kontejner ili stavke (*items*) u bazi:



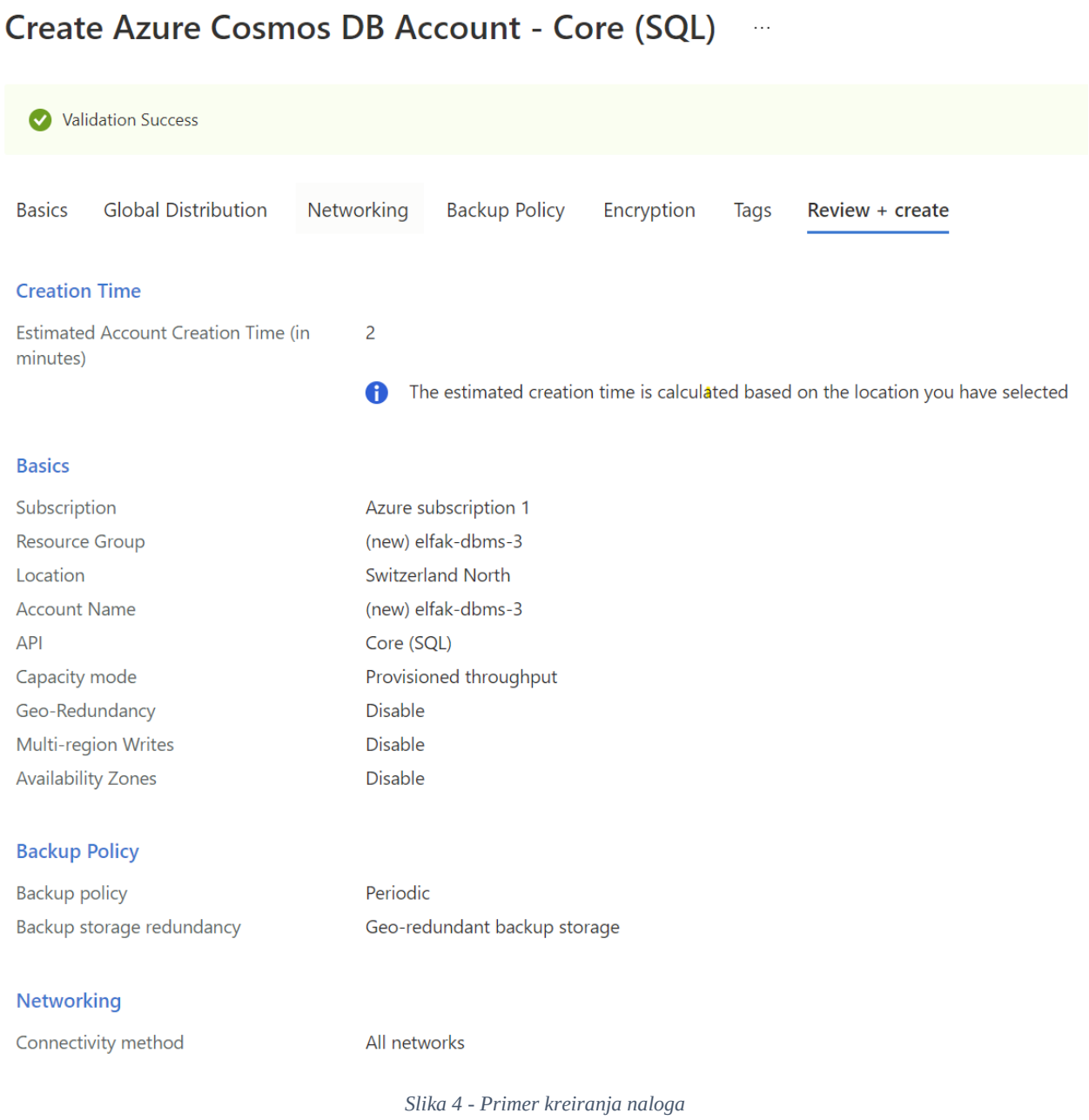

*Slika 3 - Ilustracija modela resursa*

## 4.1 Azure Cosmos Account

Nalog je osnovna jedinica globalne distribucije i visokog nivoa dostupnosti. Sadrži jedinstveno DNS ime i njime se može upravljati pomoću Azure portala, Azure CLI-a ili nekoliko SDK-ova za različite jezike.

Za globalno distribuiranje podataka i protoka se u bilo kom trenutku mogu dodati i ukloniti Azure regioni sa naloga. Nalog može i da se konfiguriše tako da ima jedan ili više regiona za upis, a može se i definisati podrazumevani nivo konzistentnosti naloga.

Trenutno se može kreirati najviše pedeset Azure Cosmos naloga sa jednom pretplatom, ali se ovo ograničenje može promeniti slanjem zahteva podršci. Jedan Azure Cosmos nalog može virtuelno da upravlja neograničenom količinom podataka i predviđenim protokom. Da bi se ovo izvelo kako treba, moguće je pod Azure nalogom kreirati jednu ili više baza podataka, a u okviru njih se može kreirati jedan ili više kontejnera.




*Slika 4 - Primer kreiranja naloga*

## 4.2 Azure Cosmos Database

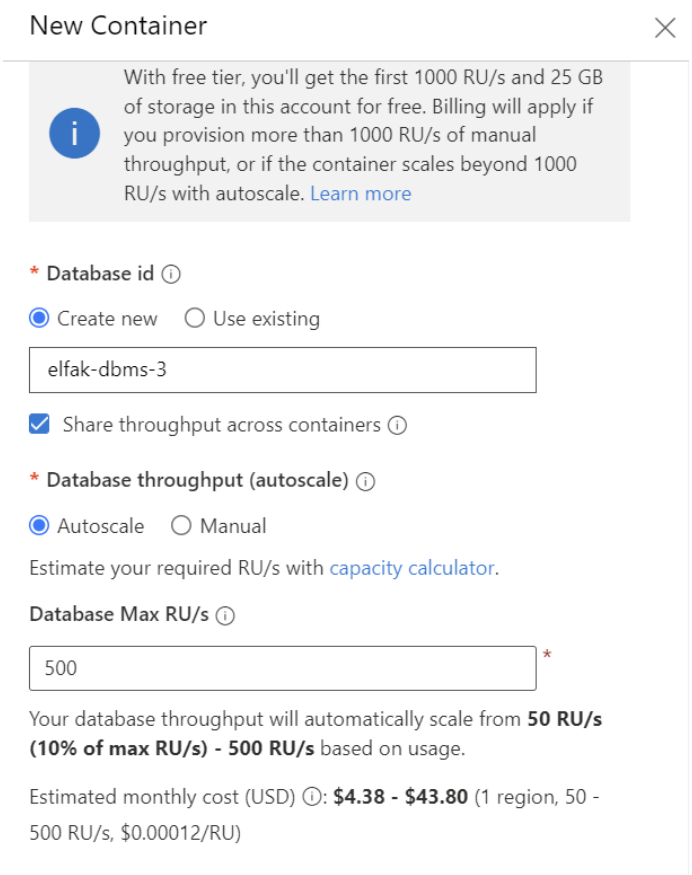
Azure Cosmos baza podataka je analogna *namespace* konceptu, odnosno predstavlja jedinicu upravljanja za skup kontejnera.

U zavisnosti od tipa API-ja, baza podataka se na sledeći način mapira u entitete:

| Azure Cosmos entity | SQL API | Cassandra API | Azure Cosmos DB API for MongoDB | Gremlin API | Table API |
|---|---|---|---|---|---|
| Azure Cosmos database | Database | Keyspace | Database | Database | NA |

> **Note**
>
> With Table API accounts, when you create your first table, a default database is automatically created in your Azure Cosmos account.

*Slika 5 - Mapiranje između baze i API-specifičnih entiteta*

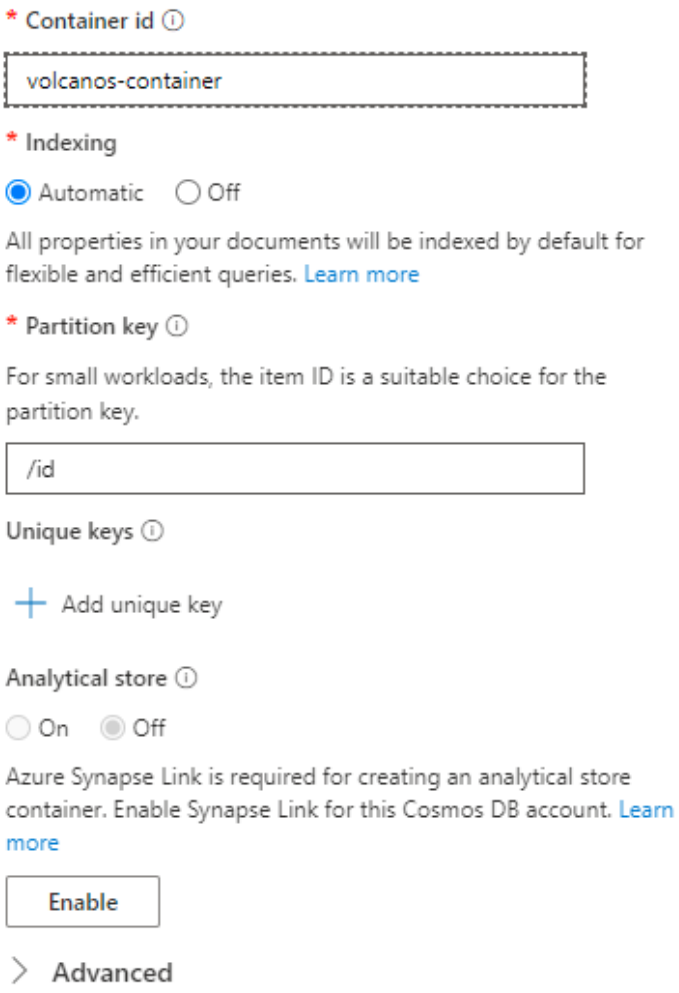



*Slika 6 - Kreiranje test baze*

## 4.3 Azure Cosmos Container

Nalog Cosmos kontejner je osnovna jedinica skalabilnosti za planirani protok podataka. Azure transparentno particioniše kontejner, koristeći logički ključ. Zahvaljujući ovom ključu, moguće je skalirati predviđeni protok i veličinu skladišta. Kontejner je horizontalno particionisan, a zatim repliciran između više regiona. Ono što se dodaje u kontejner se automatski grupiše u logičke particije, koje se distribuiraju na fizičke particije na osnovu ključa za particionisanje. Protok za kontejner je ravnomerno raspoređen po fizičkim participama.

Moguć je praktično neograničeni protok (**RU** - *Request Units per Second*) i veličina skladišta u okviru kontejnera. Kada se kreira kontejner, protok se konfiguriše na jedan od sledećih načina kao *dedicated provisioned throughput mode* gde kontejner ima garanciju na obezbeđivanje rezervisane mere protoka ili kao *shared provisioned throughput mode* u kome više kontejnera koji su deo iste baze dele rezervisani protok. Elastično skaliranje je dostupno sa oba tipa konfiguracije protoka.




*Slika 7 - Kreiranje test kontejnera*

Kontejner je agnostičan po pitanju šeme. Sve stavke (*items*) unutar kontejnera se podrazumevano automatski indeksiraju bez potrebe za eksplicitnim upravljanjem indeksom ili šemom, ali se ovo može prilagoditi dodatnom konfiguracijom politike indeksiranja na kontejneru.

Nad stavkama se mogu izvoditi *read, insert, upsert, replace, delete* operacije. Mogući formati stavki su prikazani u sledećoj tabeli:

| Cosmos entity | SQL API | Cassandra API | Azure Cosmos DB API for MongoDB | Gremlin API | Table API |
|---|---|---|---|---|---|
| Azure Cosmos item | Item | Row | Document | Node or edge | Item |

*Slika 8 - Format stavki u zavisnosti od izabranog API-ja*

Na određenim stavkama u kontejneru ili celom kontejneru se može definisati predviđeni životni vek (TTL, *time to live*), nakon čega se one brišu iz sistema i više ne vraćaju u upitima.

Opcija *Change Feed* se koristi kako bi se pravio zapis svih operacija za svaku logičku particiju kontejnera. Na ovaj način je dostupna evidencija svih ažuriranja na kontejneru, zajedno sa stanjem stavki pre i posle ažuriranja.

Za kontejner se mogu registrovati *stored* procedure, trigeri ili korisnički definisane funkcije.

Za potrebe ovog rada će se koristiti Azure Sample Data primer koji se odnosi na kolekciju vulkana u svetu, gde jedak dokument ima ovakvu strukturu:


```
{
  "VolcanoName": "Acatenango",
  "Country": "Guatemala",
  "Region": "Guatemala",
  "Location": {
    "type": "Point",
    "coordinates": [
      -90.876,
      14.501
    ]
  },
  "Elevation": 3976,
  "Type": "Stratovolcano",
  "Status": "Historical",
  "Last Known Eruption": "Last known eruption in 1964 or later",
  "id": "a6297b2d-d004-8caa-bc42-a349ff046bc4"
}
```

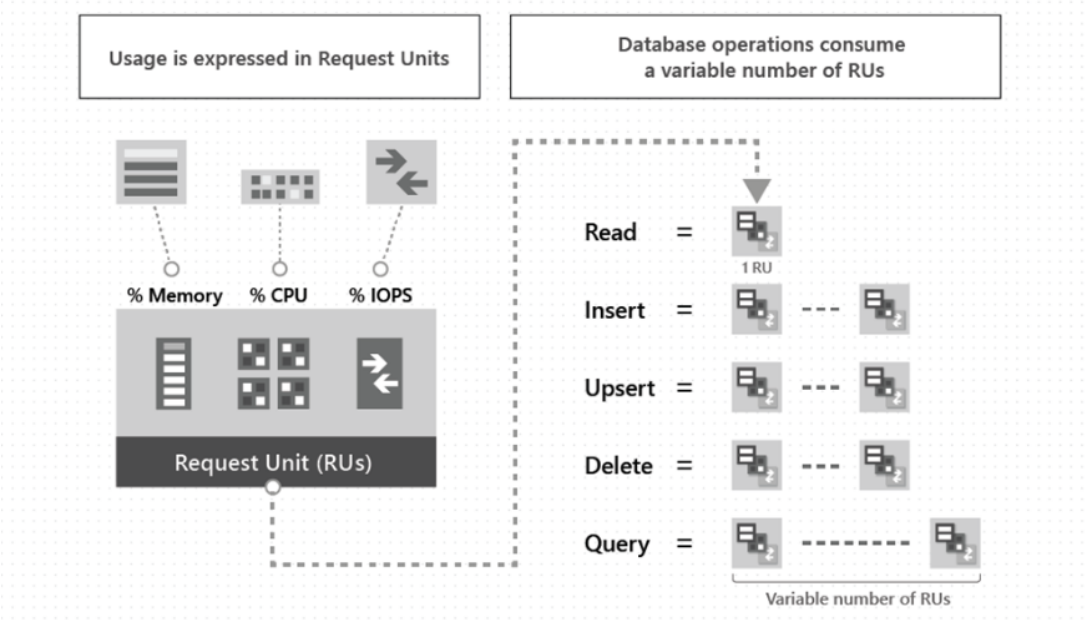

*Slika 9 - Primer dokumenta iz test kolekcije "volcanoes"*

## 4.4 Request Unit

Svaki API podržan od strane Cosmos DB servisa poseduje sopstveni skup operacija nad bazom, kod kojih je moguća značajna varijacija u zauzeću sistemskih resursa u zavisnosti od kompleksnosti operacije. Cosmos DB u te svrhe koristi normalizovanu jedinicu za cenu operacija nad bazom podataka.

*Request Unit* (**RU**) predstavlja način za upravljanje protokom, kako bi se postigle najbolje moguće performanse i može se posmatrati kao valuta za protok.

Nije potrebno pojedinačno voditi računa o resursima poput memorije ili procesorskog vremena jer je sve obuhvaćeno jednom ovakvom jedinicom. 1 RU je ekvivalentan ceni pribavljanja jedne stavke veličine 1 KB na osnovu identifikatora i vrednosti particionog ključa, i to važi za svaki podržani API.

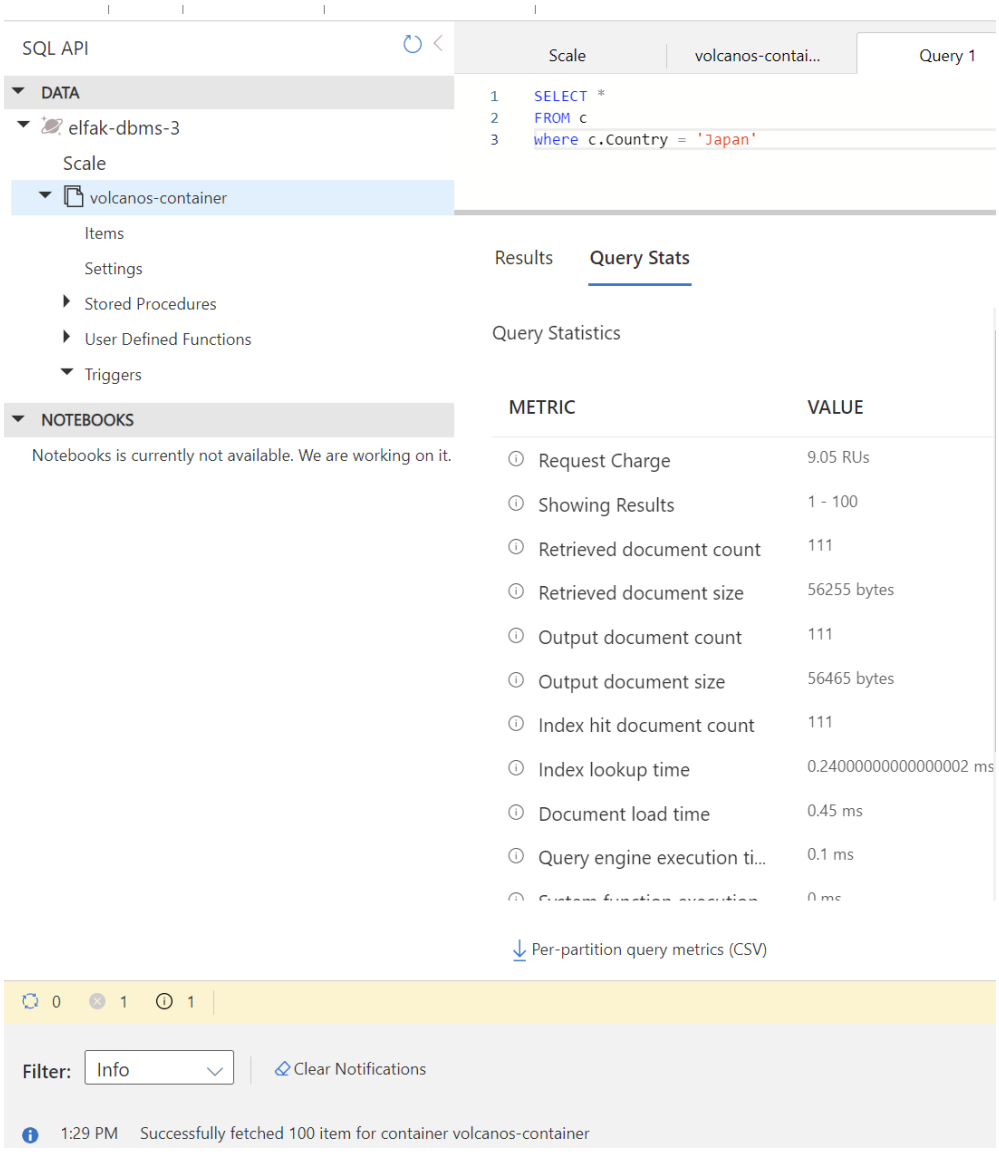



*Slika 10 - Ilustracija ideje RU-a*

Svaki zahtev se naplaćuje na ovaj način, ali nisu svi zahtevi isti u pogledu resursa, pa će se i broj RU koji se koristi razlikovati od zahteva do zahteva. Upis će uvek koštati više od čitanja jer upis ažurira više replika, a čitanje se obavlja korišćenjem samo jedne. Takođe, jednostavniji upiti će uglavnom koštati manje od složenih.

Za svaki zahtev, Cosmos DB u odgovoru šalje tačno koliko se RU-a iskoristilo. Dobra strana je to što je korišćenje ovih jedinica deterministički – uvek će identični zahtevi iskoristiti isti broj RU-a (ukoliko nije došlo do promene podataka).

Na primer, upit na slici 11 jer iskoristio 9.05 RU-a. Cena bi bila veća za složeniji upit. Na portalu su dostupni i mnogobrojni dijagrami, koje je moguće prilagoditi i koji mogu prikazivati metrike vezane za prostor, dostupnost, latenciju i konzistentnost.




*Slika 11 - Primer upita sa statistikama*

U okviru statistika upita se pored cene može videti i broj stavki, korišćenje indeksa, vreme izvršavanja, a postoji i mogućnost za generisanje detaljnih dijagrama vezanih za određene upite.

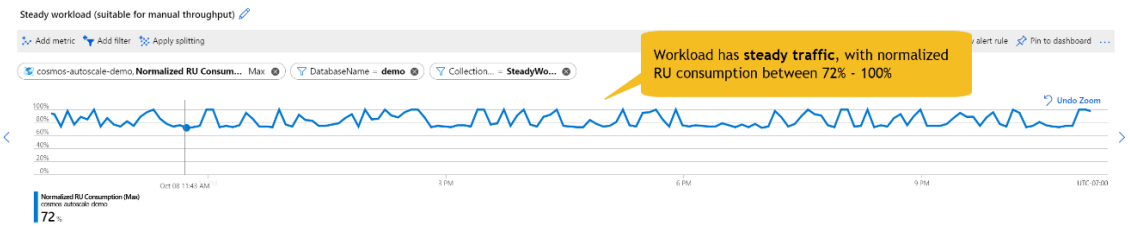
Faktori koji povećavaju potrošnju RU su npr. veličina stavke, broj njenih atributa, postojanje indeksa nad stavkom i atributima, stroga konzistentnost podatak pri čitanju, upit umesto pojedinačnog čitanja, broj i veličina rezultata upita, broj i kompleksnost predikata upita, itd.

## 4.5 Konfiguracija protoka

Azure Cosmos DB nudi mogućnosti obezbeđenog (*provisioned*) protoka na nivou baza i kontejnera. Postoje tri glavna tipa – standardni, koji se ručno podešava, *autoscale* varijanta gde su dozvoljene varijacije na standardni protok, i *serverless* režim koji u potpunosti radi na principu „plati koliko potrošiš“.

### *4.5.1 Tipovi konfiguracije protoka*

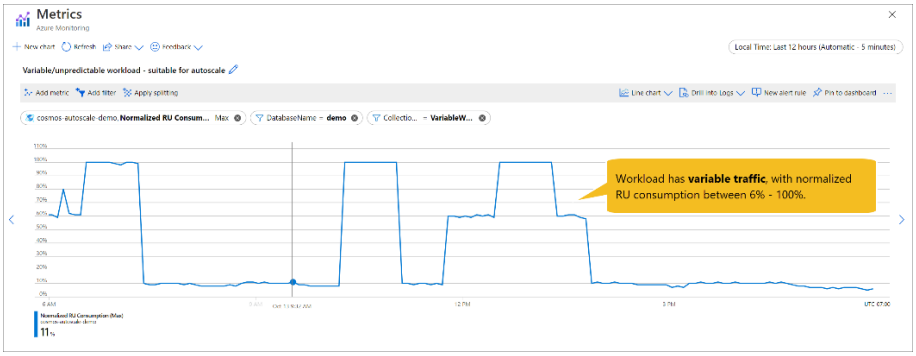
***Manual** Provisioned Throughput* – Ovom opcijom se rezerviše fiksna vrednost za RU/s koja će garantovano biti dostupna u bilo kom momentu, i ovo podešavanje se može u bilo kom trenutku ručno prilagoditi trenutnim potrebama sistema.




*Slika 12 - Stabilno opterećenje*

Pogodno je kada je opterećenje sistema stabilno i relativno predvidivo, te je moguće i automatizovati ove promene skriptama ukoliko je potrebno. Azure Cosmos DB podržava Azure PowerShell kao jednu od mogućnosti za ovakve ciljeve, i obezbeđuje i SDK za više programskih jezika.

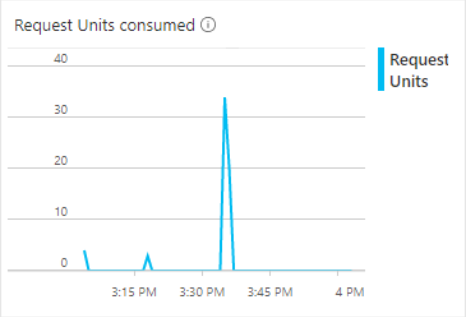
***Automatic** Provisioned Throughput* – Ova opcija automatski skalira protok sa određenim definisanim rasponom za korišćene RU/s, uz garantovanu dostupnost resursa. Skaliranje se vrši od 10% konfigurisane vrednosti do te same vrednosti koja je maksimum.




*Slika 13 - Promenljivo opterećenje*

Ova opcija je pogodna za aplikacije sa nepredvidivim ili promenljivim opterećenjima, za aplikacije za koje je bitna garantovana dostupnost, za retko korišćene aplikacije ili razvojna i test okruženja ili za unapred planirane upite.

***Serverless*** – Ova opcija se preporučuje kod sistema kod postoje nagli i veliki skokovi u opterećenju a kontejner je veći deo vremena neaktivan. To je zbog toga što se plaća samo onoliko RU koliko se iskoristi a u suprotnom nema troškova za startovanje kontejnera.




*Slika 14 - Opterećenje u viduređih naglih naleta*

Ovo podrazumeva određena ograničenja za ove tipove kontejnera, kao što je maksimum 50GB prostora za skladištenje i ograničenje naloga na jedan region.

***Burst capacity*** **–** Jedna od novih funkcionalnosti na Azure Cosmos DB bazi je mogućnost akumuliranja kapaciteta za protok tokom mirovanja baze odnosno kontejnera. Ovime se zahtevi koji bi inače bili odbačeni mogu opslužiti dok je ovaj „kapacitet za nalet“ dostupan. Ova mogućnost je dostupna samo kod konfiguracija sa obezbeđenim protokom do 3000 RU/s, i nije dostupna za *serverless* režim.

Najjednostavniji primer bi bila fizička particija sa 100 RU/s obezbeđenog protoka koja miruje 5 minuta. Za to vreme ona akumulira 100 RU/s * 300 sekundi = 30000 RU kapaciteta za nalet. Budući da se u slučaju naglog porasta broja zahteva dopušta do 3000 RSU, ovo znači da particija može ovako značajno pojačati svoj protok do 30000 / 3000 = 10 sekundi. Ovi zahtevi pri standardnoj konfiguraciji od 100 RU/s bili odbačeni sa HTTP statusnim kodom 429 (*too many requests*).

### *4.5.2 Granularnost konfiguracije protoka*

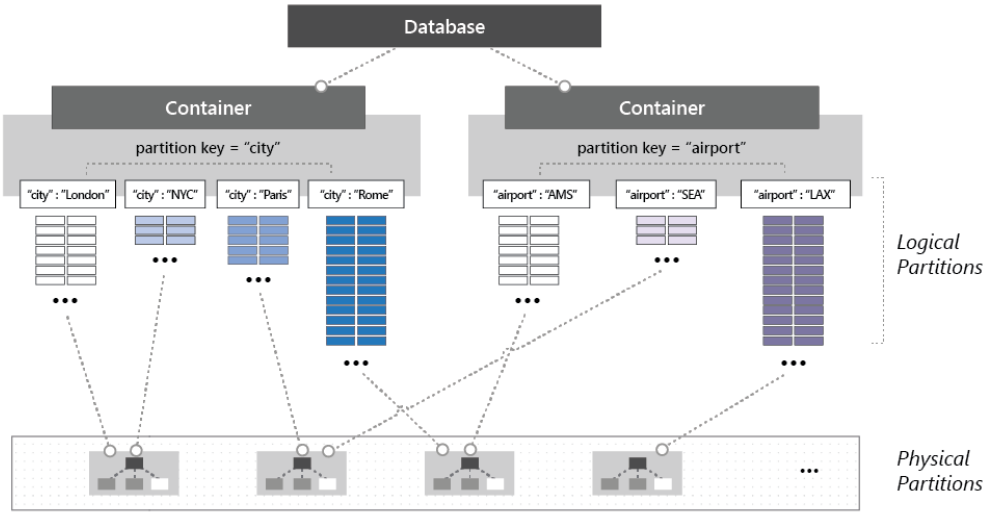
**Protok obezbeđen na nivou kontejnera** je rezervisan isključivo za taj kontejner i dostupan mu je sve vreme. Ovo je podrazumevana opcija prilikom podešavanja i najčešće korišćen slučaj. Protok se ravnomerno raspoređuje po fizičkim particijama, a ako je particioni ključ dobro izabran, biće ravnomerno raspoređen i po logičkim particijama.

Protok se ne može dodeliti određenoj logičkoj particiji, i budući da se protok deli po fizičkim particijama, može doći do ograničenja protoka ako se je neka od logičkih particija nesrazmerno opterećena. Ova pojava se naziva „vruća particija“ (*hot partition)* i biće detaljnije opisana u narednom poglavlju na temu particionisanja.

**Protok obezbeđen na nivou baze** je deljen između svih kontejnera unutar baze, sa izuzetkom kontejnera koji imaju sopstveni obezbeđeni protok. Ovaj princip je ekvivalentan bazi postavljenoj na klasteru računara/virtuelnih mašina.

Ne može se garantovati predvidljiv protok na nivou pojedinačnog kontejnera, je protok za kontejner zavisi od njihovog ukupnog broja, izbora particionih ključeva i distribucije opterećenja svih logičkih particija.

Ovaj pristup se preporučuje kada nije naročito bitan protok na nivou određenog kontejnera. Ista ograničenja opisana za logičke particije se primenjuju i na nivou ove podele.




*Slika 15 – Primer raspodele logički i fizičkih particija na primeru dva kontejnera unutar jedne baze.*

Na slici iznad se vidi primer gde jedna fizička particija može biti odgovorna za samo jednu logičku particiju ili za više logičkih particija, čak i iz različitih kontejnera. Zbog toga je jako bitno odraditi dobro particionisanje.

# 5. Osnove Azure Cosmos DB NoSQL sistema

Azure Cosmos DB NoSQL API pruža poznati način pristupa bazi putem SQL upita dajući privid klasične relacione baze, dok su podaci zapravo skladišteni bez specificirane šeme u JSON formatu.

Podacima se može pristupati direktno na nivou stavke (*point-read*) pomoću pretrage po ključu ili putem upita koji se oslanjaju na indekse. Svi podaci unutar baze se automatski indeksiraju preko sistema koji je nezavisan od šeme.


```
[
  {
    "Volcano Name": "Kilauea",
    "Country": "United States",
    "Region": "Hawaiian Is",
    "Location": {
      "type": "Point",
      "coordinates": [
        -155.292,
        19.425
      ]
    },
    "Elevation": 1222,
    "Type": "Shield volcano",
    "Status": "Historical",
    "Active": true,
    "Last Known Eruption": "2021-09-29",
    "id": "27222e2d-1478-8b75-eb09-08073fcb175a"
  },
  {
    "Volcano Name": "Leskov Island",
    "Continent": "Antarctica",
    "Region": "Antarctica",
    "Location": {
      "type": "Point",
      "coordinates": [
        -28.13,
        -56.67
      ]
    },
    "Elevation": 190,
    "Type": "Stratovolcano",
    "Status": "Fumarolic",
    "Active": false,
    "Last Known Eruption": "Undated, but probable Holocene eruption",
    "id": "dc4afd1a-a51a-d2ba-e604-112d10b4fc5a"
  }
]
```


*Slika 16 - Primer podataka vezanih za vulkane*

Na slici 16 se vidi primer podataka vezanih za vulkane u svetu. Entiteti u nizu sadrže brojčane, tekstualne, logičke i vremenske podatke, proizvoljno organizovanih u strukturu stabla i nizova, gde neki imaju polje “Country” a neki “Continent”.

Upiti takođe proizvode JSON, gde je moguće primeniti projekcije po kolonama, postavljati većinu poznatih SQL predikata i obavljati JOIN operaciju nad stavkama. Budući da ne postoji šema, tipovi podataka se određuju dinamički. Upit može vratiti objekte istog imena ali različitog tipa.

Na primeru sledećeg upita se može videti kako se može birati projekcija i postavljati predikat:


*Slika 17- Primer upita*

Svaka stavka se unutar kontejnera skladišti u obliku stabla. Za svaku stavku se kreira preudo-koreni čvor koji sadrži sva polja na prvom nivou (“Volcano Name”, “Region”, “Location”…). Takođe se za svaki element niza kreira međučvor označen rednim brojem tog čvora u nizu.

Azure Cosmos DB ovime omogućava poljima da budu referencirana na osnovu njihove putanje unutar stabla. Neka od polja za stavke iznad se mogu referencirati pomoću sledećih putanja:

- /Active: false
- /Location/type: “Point”
- /Location/coordinates/0/28.13


*Slika 18 - Podrazumevana podešavanja za kreiranje indeksa*

Na slici 18 se mogu videti podešavanja za indeksiranje. Jedino polje koje se podrazumevano ne indeksira je *_etag* koje označava verziju stavke i koristi se prilikom operacija upisa. Podržani su i geospatijalni indeksi.

**TTL** - Još jedna bitna osobina kontejenera i stavki u njemu je što je nad njima moguće definisati period nakon koga će biti automatski obrisane, odnosno *Time to Live* (TTL). Ove operacije brisanja konzumiraju RU u pozadini koji nisi bili iskorišćeni tokom upita.

U slučaju da je kontejner preopterećen zahtevima, brisanje će biti odloženo dok ne bude dovoljno kapaciteta protoka. Podaci se više ne vraćaju u upitima u slučaju da im je TTL istekao, čak iako još uvek nisu stvarno obrisani.

***Unique Key Constraint* –** Azure Cosmos DB omogućava određenu dozu integriteta podatak na nivou logičke particije. Moguće je postaviti ograničenje jedinstvenog ključa, gde će biti sprečeno dodavanje ili izmena stavki koje bi prouzrokovale dupliranje polja određenog kao jedinstveni ključ.

Ovo ograničenje se podešava prilikom kreiranja kontejnera i ne može biti promenjeno u drugačiji ključ kasnije. Takođe prouzrokuje veću potrošnju RU. Takođe treba obratiti pažnju na to da je prisustvo ključa obavezno – nedostatak ključa se tretira kao *null* vrednost, koja ulazi u obzir kao jedinstvena vrednost, te sme postojati samo jedna stavka bez ključa na nivou kontejnera.

**Transakcije –** Azure Cosmos DB podržava ACID transakcije sa „snapshot“ nivoom izolacije na nivou logičke particije kontejnera. Operacije se izvršavaju unutar replike particije i moguće je vršiti i čitanje i upis pod transakcijom.

Budući da Cosmos DB podržava procedure, triggere i funkcije, moguće je pisati naprednu logiku u nativno podržanom JavaScript okruženju, što omogućava moćan sistem za kontrolu toka i nošenje sa greškama prilikom pisanja ovih operacija. *Exception* u JavaScript-u automatski znači poništenje transakcije. Prednost je, kao kod MongoDB baze, što nema problema nepoklapanja između modela u bazi i u kodu, jer imamo JavaScript/JSON system.

Transakcije se izvršavaju sa optimističnom kontrolom konkurentnosti. Svaka stavka ima interno polje *_etag* koje se automatski generiše na serveru svaki put kada se upisuje u stavku. Ovo omogućava serveru da na osnovu zaglavlja u zahtevu uporedi da li se verzija stavke koju klijent ima poklapa sa aktuelnom verzijom stavke, i da HTTP statusnim kodom 412 odgovori na zahtev u suprotnom. Klijent nakon toga može osvežiti lokalnu kopiju stavke i ponoviti zahtev.

Ovo ponašanje transakcije je funkcionalno na nivou pojedinačnog regiona. U slučaju da se u više različitih regiona upisu različiti podaci za istu stavku, biće pokušano spajanje. Ukoliko konflikti ne mogu da se razreše, transakcije će biti poništena.

# 6. Particionisanje

Relacione baze podataka prilikom rasta zahtevaju povećanje procesorske moći i skladišnog prostora na računaru ili virtuelnoj mašini na kojoj su smeštene.

Za razliku od ovakvog vertikalnog skaliranja, kod NoSQL baza kao što je Azure Cosmos DB imamo pristup zasnovan na horizontalnom skaliranju, gde se prilikom rasta baze povećava broj samih servera, odnosno particija, umesto unapređivanja pojedinačnih servera.

Ovaj koncept nam omogućava masovno skaliranje baze podataka u pogledu prostora i protoka. Samim kreiranjem kontejnera, Cosmos *DB* particioniše podatke koji se čuvaju u kontejneru i na taj način upravlja njihovim porastom.Ono što je na površini je kontejner, kao jedini logički resurs u kome se čuvaju podaci i šalju upiti. Kontejner može da raste i skladišti sve više i više podataka i nije potrebno brinuti o skaliranju.

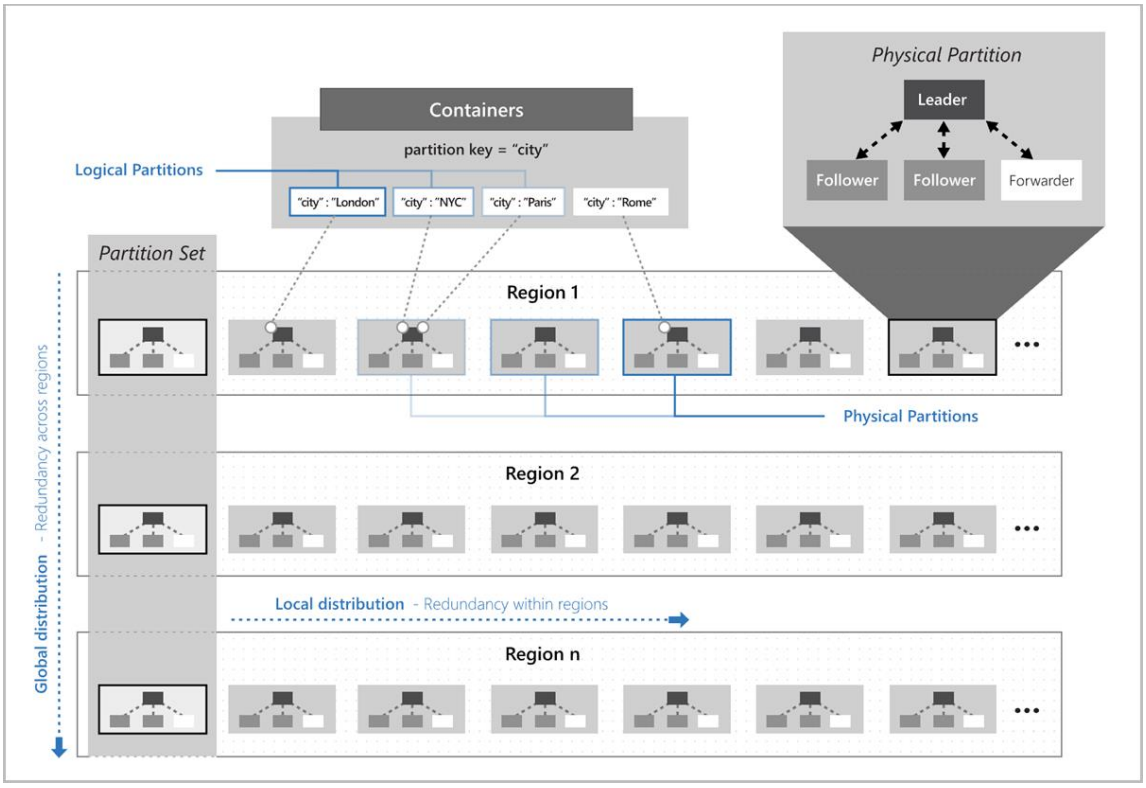
Da bi se podacima raštrkanim na različitim particijama efikasno pristupalo, potrebno je da se efikasno ih organizovati. Za svaki kontejner je potrebno definisati **particioni ključ** (*partition key*), koji se specificira kao putanja do JSON polja prema čijoj vrednosti se stavke raspoređuju u zasebne logičke particije.

Dokumenti sa istom vrednošću ključa se uvek smeštaju u istu **logičku particiju**. Particioni ključ može biti neki deljeni atribut stavki ili zaseban identifikator iz koga sledi da svaka stavka pripada sopstvenoj logičkoj particiji. Uglavnom je dobro izabrati ključ sa što širim skupom vrednosti, jer u suprotnom particija može narasti i do 20 GB, koliko je maksimalna dozvoljena vrednost.

Kontejner se skalira distribuirajući podatke i protok na više **fizičkih particija**, na koju interno može biti mapirana jedna ili više logičkih particija. Manji kontejneri tipično sadrže veliki broj

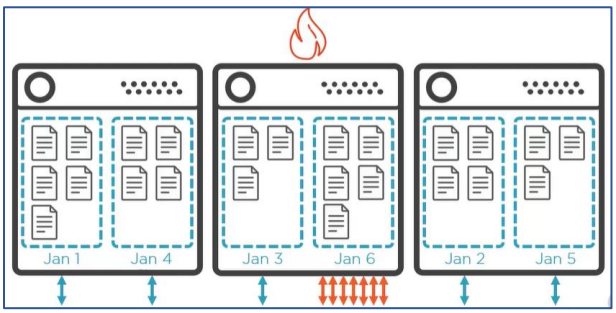
logičkih particija smeštenih na jednu fizičku particiju. Ovo je interni implementacioni detalj Azure Cosmos DB sistema i ne može se podešavati od strane korisnika.

Fizičke particije se kreiraju i dele u skladu sa podacima i protokom i za njih nema brojnog ograničenja. Deljenje particije ne utiče na dostupnost aplikacije, jer ono samo kreira novo mapiranje logičke na fizičku particiju, bez uticaja na staro mapiranje. Svaka od njih je distribuirana na najmanje 4 replike u sklopu jednog data centra.



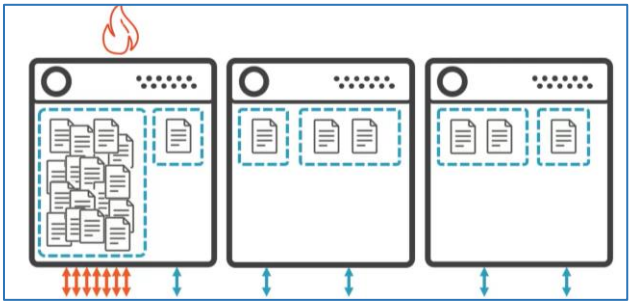

*Slika 19 - Šema particionisanja*

Ako se ne izvrši dobar izbor particionog ključa, može doći do nejednake raspodele protoka među fizičkim particijama, što može dovesti do ranije pomenute „vruće particije“.




*Slika 20 - Hot partition za slučaj particija sa neravnomernim protokom*

Vruća particija može biti i posledica toga da su određene logičke particije nesrazmerno velike u odnosu na ostatak što onemogućuje dobar model skaliranja u sistemu.



*Slika 21 - Hot partition za slučaj nesrazmerno velike particije*

Cosmos DB u pozadini kreira onoliko particija koliko je potrebno u skladu sa podacima. Particije predstavljaju individualne fizičke jedinice fiksnog kapaciteta, pri čemu svaka od njih predstavlja i jedinicu replikacije. Što znači da je svaka fizička particija kontejnera distribuirana na više replika (najmanje četiri u sklopu jednog data centra), kako bi bila obezbeđena visoka dostupnost podataka.

Azure Cosmos DB poseduje koncept **sintetičkog ključa** i **hijerhijskog particionisanja** koje može biti izutetno korisno u slučajevima koji se ne uklapaju u potpunosti ni u jedan od gorepomenutih preporuka za izbor particionog ključa.

Ako imamo *multi-tenant* aplikaciju, particionisanje na nivou tenanta će imati kao rezultat ogromne particije dok će korišćenje korisničkog identifikatora dovesti do toga da svaki upit bude *cross-partition.* Moguće je definisati sintetički ključ koji kombinuje *TenantId i UserId*, pa ići čak i nivo niže do polja kao što je npr. *SessionId*, sa tim što je dozvoljeno najviše tri nivoa dubine.

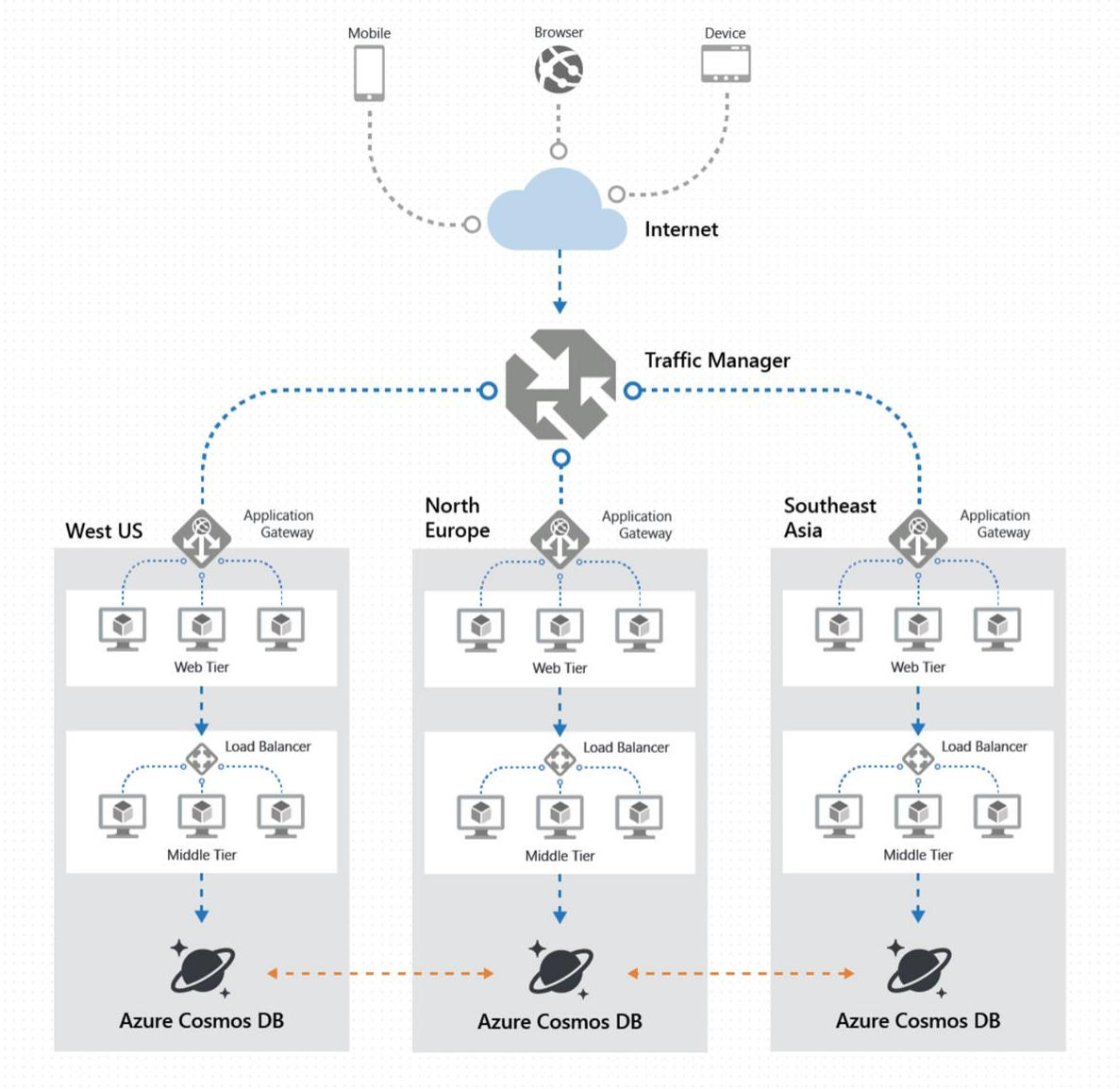
Ovime će upiti koji specificiraju ova polja biti usmereni na podskup fizičkih particija koji sigurno sadrže tražene podatke. Npr za kontejner koji sadrži 1000 fizički paritcija gde se podaci vezani za određenog tenanta nalaze na samo njih 5, obim podataka koje upit treba pretražiti se smanuje čak 200 puta.

# 7. Globalna distribucija

Globalna distribucija podatak je ključna strategija u postizanju niske latencije i visoke dostupnosti aplikacija, i podrazumeva da se se podaci nalaze geografski blizu korisnika i replikuju u svim ostalim data centrima u svetu.

Još jedna prednost je lak oporavak od kvarova. Ako određena replika otkaže na fizičkoj particiji, postoje druge koje će obezbediti da kontejner nastavi da radi bez prestanka, što kod globalne

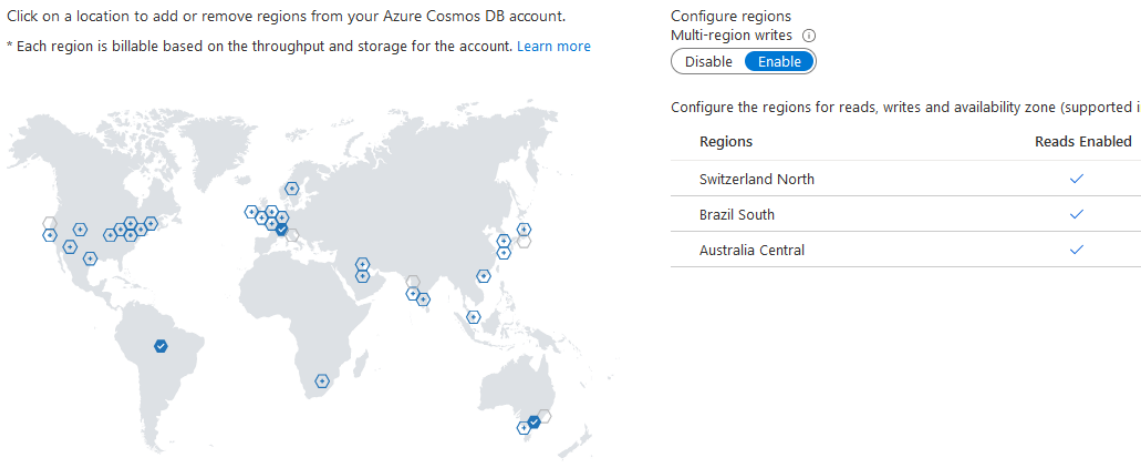
replikacije podrazumeva dostupnost podataka čak i u slučaju otkazivanja infrastrukture celog regiona, npr. zbog prirodnih nepogoda.




*Slika 22 - Topologija Azure Cosmos DB sistema*

Prilikom kreiranja naloga se bira inicijalni region, a zatim se mogu dodati svi regioni u koje je poželjna replikacija podataka jednostavnim klikom na mapi u korisničkom interfejsu. Upis podataka je podrazumevano ograničen samo na inicijalni region.

Ukoliko želimo da omogućimo upis iz više regiona, potrebno je omogućiti *multi-master* opciju. Ovo opcija pruža relativno nisko kašnjenje pri operacijama iz sekundarnih regiona.




*Slika 23 - Podešavanje georeplikacije*

Kod upisa iz više regiona može doći do konflikta ukoliko dva korisnika iz različitih regiona ažuriraju isti dokument u isto vreme, pre nego što sistem uspe da geo-replicira podatke

Postoje dve glavne opcije:

- ***Last Write Wins* (LWW) –** Podrazumevana opcija za rešavanje konflikta koristeći sistemsko polje sa *timestamp*-om poslednjeg upisa *_ts*, upotrebom protokola za sinhronizaciju časovnika. Ako se koristi NoSQL API, moguće je specificirati bilo koje numeričko polje za poređenje odnosno ***conflict resolution path***. U slučaju konflikta će stavka sa većom vrednošću ovog polja pobediti i prepisati drugu stavku, dok sistem određuje pobednika u slučaju iste vrednosti. Operacija brisanja uvek ima primat nad upisom.
- ***Custom* –** Ovaj sistem prepušta razrešenje konflikta aplikativnoj semantici. Neophodno je registrovati **proceduru** koja će se okinuti pri detekciji konflikta u toku transakcije. Sistem garantuje da će se procedura izvršiti barem jednom. Ukoliko se ne registruje procedura ili se konflikti njome ne razreše, biće poslati u *conflicts feed* gde se moraju manuelno razrešiti.

## 7.1 Nivoi konzistentnosti

Azure Cosmos DB podržava pet nivoa konzistentnosti gde postoji obrnuta proporcionalnost između performansi i „prljavog“ čitanja (*dirty reads*).

**Strong Consistency –** Sa ovim nivoom smo sigurni da uvek čitamo poslednju verziju podataka, ali je latencija veća prilikom upisa. Kada se vrši upis, mora da se sačeka da Azure Cosmos DB potvrdi da je upis uspešno sačuvan u većini replika. Ovo garantuje konzistentna čitanja sa niskom latencijom, ali i čitanja moraju da budu potvrđena od strane većine replika pa je samim tim broj utrošenih RU veći.


*Slika 24 - Odnos nivoa konzistentnosti*

**Bounded Staleness Consistency –** Sa ovim nivoom je moguće tolerisati zastarele podatke, ali do određene granice koja se može konfigurisati. Za podatke koji prevaziđu tu granicu, *Cosmos DB* prelazi na jak nivo konzistencije. Kao i za prethodni nivo, čitanja moraju da budu potvrđena od strane većine replika.

**Session Consistency –** Ovo je podrazumevani nivo latencije. Uspostavlja se sesija, tako što se generiše jedinstveni ključ sesije i korisnik koji upisuje ga uključuje u svoje zahteve. Na ovaj način je onom ko vrši upis zagarantovano da će pročitati iste podatke koje su upisali, ali svi ostali mogu da dobiju *dirty read.* Čitanja se uglavnom obavljaju iz jedne replike i nije potreban kvorum.

**Consistent Prefix –** Sa ovim nivoom, čitanja se uvek obavljaju iz jedne replike i *dirty reads* su uvek mogući. Ukoliko se dobiju zastareli podaci, oni su već ažurirani od strane većine replika. Nikada neće doći do čitanja zastarelih podataka van redosleda. Ukoliko se neki podaci ažuriraju tri puta, jednom kada dobijemo drugu verziju podataka više nije moguće dobiti prvu najstariju verziju.

**Eventual Consistency –** Sa ovim nivoom nemamo nikakvu garanciju, ali je najniža latencija prilikom upisa i najbolji je po performansama u poređenju sa prethodnim nivoima. Ne čeka se na potvrde prilikom upisa, i podaci se dobijaju iz bilo koje replike u bilo kom trenutku. Zastareli podaci se mogu javiti u bilo kom redosledu.

# 7. Literatura

[1] Abourezq, Manar & Idrissi, Abdellah. (2016). Database-as-a-Service for Big Data: An Overview. International Journal of Advanced Computer Science and Applications. 7. 10.14569/IJACSA.2016.070124.

[2] https://docs.microsoft.com/en-us/azure/cosmos-db/