

Elektronski fakultet, Univerzitet u Nišu
Katedra za Računarstvo i informatiku

SEMINARSKI RAD

Cloud baze podataka i Database-as-a-Service
na primeru AWS DynamoDB

Predmet: Sistemi za upravljanje bazama podataka

Student: Nikola Stanisavljević
Broj indeksa: 940

Sadržaj

Uvod	2
Cloud baze podataka	3
Amazon Cloud	5
NoSQL pokret	5
Amazon Web Services	5
Amazon DynamoDB	6
Osnovne komponente DynamoDB baze	8
Particionisanje podataka	10
DynamoDB indeksi	14
DynamoDB Streams	15
Konzistentnost operacija Read/Write	16
Pristup DynamoDB	17
AWS SDK podrška za DynamoDB	23
Memorijsko ubrzanje - DAX	24
Zaključak	26
Literatura	27

Uvod

Težnja ovog rada je da prikaže osnovne načine funkcionisanja Amazon DynamoDB rešenja. Cloud baze podataka predstavljaju novi trend kroz pružanje usluga i servisa. U radu se osvrćemo na osnove cloud baza i Amazon Cloud, kao i NoSQL pokret u kome je nastala Dynamo baza, preteča DynamoDB rešenja.

DynamoDB je Amazonov predstavnik u NoSQL svetu sa svojim Key-Value i Document modelom podataka. Poseban značaj ogleda se u distribuiranom cloud sistemu i prisutnoj automatizaciji određenih procesa koja dodatno olakšava njihovu integraciju sa korisničkim aplikacijama. Veliki uspeh DynamoDB postiže svojim mogućnostima za dobre performanse, velikom brzinom, skaliranjem i replikacijom podataka u okviru AWS regiona.

U okviru rada obrađeni su tipovi podataka, strukture tabela i organizacija baze uz particionisanje i smeštanje podataka u memoriju. Prikazani su različiti načini pristupa AWS servisima, konkretno DynamoDB bazi uz praktičnu ilustraciju i primere.

Cloud baze podataka

Cloud baza podataka radi na nekoj platformi u oblaku, a pristup bazi se pruža kao usluga (*engl. as-a-service*). Usluge baza podataka vode računa o skalabilnosti (*engl. scalability*) i visokoj dostupnosti (*engl. high availability*) baze i čine osnovni paket softvera za korisnike. Postoje dve primarne metode za pokretanje baze podataka u oblaku:

- **Slika virtualne mašine (*engl. Virtual machine image*)**

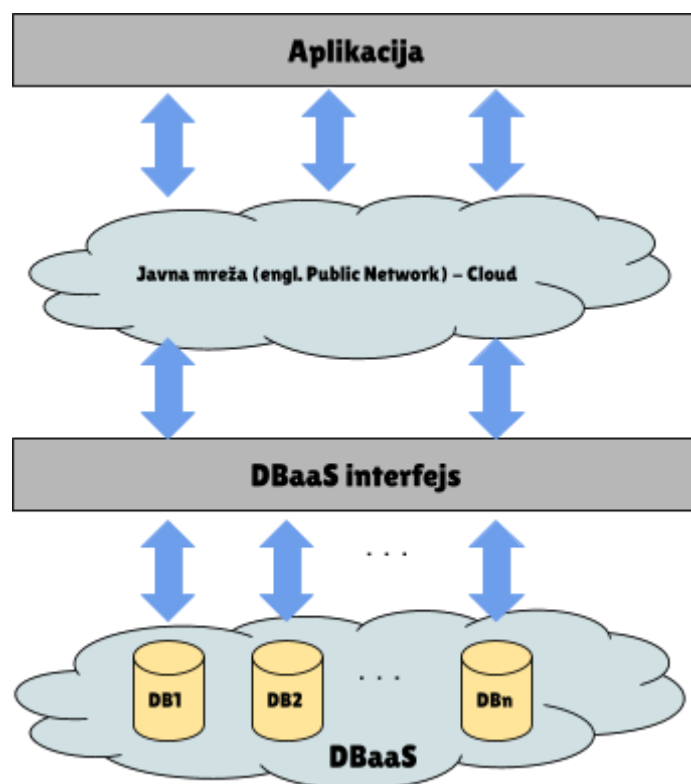
Platforma omogućava korisniku da kupi instancu virtualne mašine na ograničeno vreme, a na takvim virtualnim mašinama može se pokrenuti baza podataka. Korisnici mogu da koriste i gotove slike mašina koje već uključuju optimizovanu instalaciju baze ili da opreme sopstvenu sliku mašine sa svojim konfiguracijama.

- **Baza kao usluga (*engl. Database-as-a-service*)**

Kada je baza podataka kao model usluge, vlasnici aplikacija ne moraju sami da instaliraju i održavaju bazu podataka. Provajder baze preuzima odgovornost za instaliranje i održavanje baze, a vlasnicima aplikacija se naplaćuje korišćenje usluge. Ovo spada u SaaS model - softver kao usluga (*engl. Software-as-a-Service*).

Model podataka/Način pokretanja	VM	DBaaS
SQL	<ul style="list-style-type: none">● EDB Postgres Advanced Server● IBM DB2● Ingres● MariaDB● MySQL● NuoDb● Oracle Database● PostgreSQL● SAP HANA	<ul style="list-style-type: none">● Amazon RDS● Amazon Aurora● Alibaba AnalyticDB● Alibaba PolarDB● Clustrix● EnterpriseDB● Google Cloud SQL● Heroku● Oracle Database Cloud Service● Microsoft Azure SQL● Snowflake Cloud Data Warehouse
NoSQL	<ul style="list-style-type: none">● Apache Cassandra● ArangoDB● Clusterpoint Database Virtual Box VM● CouchDB● EDB Postgres Advanced Server● Hadoop● MarkLogic● MongoDB● Neo4j	<ul style="list-style-type: none">● Amazon DynamoDB● Azure CosmosDB● Cloudant Data Layer (CouchDB)● EnterpriseDB● Google Cloud Bigtable● Google Cloud Datasore● MongoDB● Oracle NoSQL Database Cloud Service● Amazon DocumentDB

Tabela1. Podela provajdera cloud baza podataka na osnovu načina pokretanja i modela podataka



Slika1. Arhitektura DBaaS sistema

Slika1 prikazuje arhitekturu sistema koji koristi bazu kao uslugu (*skr. DBaaS*), jedan od prethodno pomenutih tipova cloud baza podataka. DBaaS je servis koji omogućava krajnjim korisnicima, bilo da su oni deo tima, administratori baza podataka ili programeri, da veoma brzo obezbede okruženje sa bazom podataka. Ključne karakteristike baza kao usluga su:

- **Self-service** - Korisnici baze mogu biti iz različitih oblasti i sa različitim iskustvom, a veoma lako mogu koristiti bazu podataka.
- **On-demand** - Plaćanje se preračunava na osnovu korišćenih resursa.
- **Dinamičnost** - Omogućava fleksibilnu bazu podataka koja se prilagođava trenutnim potrebama okruženja na osnovu raspoloživih resursa.
- **Bezbednost** - Tim stručnjaka konstantno nadgleda servise i stoji na raspolaganju korisniku.
- **Automatizacija** - Administracija i nadgledanje baze podataka su automatizovani procesi.
- Upotrebljava postojeće servere i skladišta.

Amazon Cloud

NoSQL pokret

NoSQL je pokret koji je predstavljen 1998. godine, a obuhvata sve nerelacione baze podataka. Neki od primera *Not-Only-SQL* baza podataka su Google *BigTable* iz 2006. godine, Amazonov *Dynamo* iz 2007. godine, *Cassandra* u upotrebi Facebook-a iz 2008. godine i *Voldemort* koji koristi LinkedIn iz 2009. godine. Relacione baze podataka su korisne za efikasno skladištenje podataka, jer imaju podršku za ACID¹ transakcije i kompleksne SQL upite. Kada su u pitanju podaci na web-u i obrada tih podataka, pravila se menjaju i upotreba relacionih baza podataka gubi na značaju posebno zbog nedostatka fleksibilnosti.

Osnovna podela NoSQL baza podataka data je u nastavku sa njihovim karakterističnim predstavnicima u zagradama:

- Key/Value Stores (*Dynamo, Redis, Voldemort*)
- Column stores (*BigTable, Cassandra*)
- Document stores (*MongoDB*)
- Graph databases (*Neo4j*)

Amazon Web Services

Amazon Web Services (skr. AWS) je deo kompanije Amazon koji pruža platforme i API²-je za korišćenje u oblaku na zahtev pojednicima i kompanijama po modelu plaćanja servisa po upotrebi (*engl. pay-as-you-go*). Zajedno, ovi web servisi za računarstvo u oblaku pružaju skup primitivne apstraktne tehničke infrastrukture i distribuiranih računarskih razvojnih blokova i alata. AWS-ova verzija virtualnih računara oponaša većinu atributa realnih računara, uključujući hardverske procesore (CPU i GPU), RAM memoriju, hard disk ili SSD, izbor operativnih sistema, umrežavanje, baze podataka, upravljanje odnosa sa klijentima (*skr. CRM*) i druge.

AWS sadrži više od 212 servisa uključujući računare, skladišta, umrežavanje, baze podataka, analitiku, aplikativne usluge, deployment, upravljanje, alate za programiranje i IoT³. Najpopularnije su EC2 (*engl. Amazon Elastic Computer Cloud*) i Amazon S3 (*engl. Amazon Simple Storage Service*). Mnoge usluge nisu direktno izložene krajnjim korisnicima, već nude funkcionalnosti putem API-ja koje programeri mogu koristiti u svojim aplikacijama.

¹ *ACID* - *Atomicity Consistency Isolation Durability* - predstavlja skup svojstava transakcije u bazama podataka namenjenih da garantuju validnost čak i u slučaju grešaka, nestanka struje i slično.

² *API* - *Application Programming Interface* - predstavlja interfejs za programiranje koji definiše načine na koje aplikacije mogu da zahtevaju usluge od biblioteka i/ili operativnih sistema.

³ *IoT* - *Internet of Things* - predstavlja međumrežavanje fizičkih uređaja i različitih senzora i povezivanje u cilju razmene podataka sa drugim uređajima, proizvođačem i/ili operaterima.

Uslugama se pristupa putem HTTP⁴ zahteva, koristeći REST⁵ arhitekturni stil i SOAP⁶ protokol za starije API-je i isključivo JSON⁷ za novije verzije.

Amazon DynamoDB

Amazon DynamoDB predstavlja NoSQL Amazon uslugu koja funkcioniše kao Key-Value i Document baza podataka. Zahteva samo primarni ključ i nije potrebno imati definisanu šemu za pravljenje tabele. Može da skladišti bilo koju količinu podataka i podrži bilo koju količinu saobraćaja. Upotrebom DynamoDB baze očekuju se dobre performanse čak i sa skaliranjem.

Neke od kritičnih funkcija upravljanja DynamoDB bazom su sledeće:

- Automatska replikacija podataka na preko 3 zone dostupnosti u jednom regionu;
- Neograničeno skaliranje za operacije čitanja i upisa. Ako se saobraćaj povećava DynamoDB povećava propusnost i prilagođava se opterećenju i obrnuto.
- Sigurnosne kopije (*engl. backup*) na Amazon S3;
- Integracija sa drugim AWS servisima kao što su Elastic MapReduce (skr. EMR), Data Pipeline i Kinesis;
- Pay-per-use-model naplaćivanje se vrši na osnovu količine podataka i upotrebe servisa;
- Bezbednost i kontrola pristupa regulisane su putem IAM servisa (*engl. Identity and Access Management*);
- Enterprajz funkcionalnosti kao što su monitoring alati, privatni VPN (*engl. Virtual Private Network*) i robustan SLA (*engl. Service Level Agreement*).

Ove funkcionalnosti DynamoDB baze mogu se primeniti u sledećim kategorijama:

- Režim kapaciteta na zahtev (*engl. on-demand capacity mode*) - Aplikacije koje koriste ovaj servis imaju omogućeno automatsko skaliranje kako bi se prilagodile odgovarajućim uslovima odnosno radnom opterećenju;
- Ugrađena podrška za ACID transakcije na serverskoj strani;
- Režim sigurnosne kopije na zahtev (*engl. on-demand backup*) omogućava kreiranje kompletnih rezervnih kopija koja ne utiče na performanse ili dostupnost trenutnih aplikacija;

⁴ HTTP - HyperText Transfer Protocol - predstavlja mrežni protokol, jedan od internet protokola koji je glavni i najčešći metod prenosa informacija na web-u.

⁵ REST - Representational State Transfer - arhitektonski stil softvera koji definiše skup ograničenja koja će se koristiti za kreiranje web usluga.

⁶ SOAP - Simple Object Access Protocol - specifikacija protokola za razmenu poruka koje čine strukturane informacije u implementaciji web usluga računarskih mreža.

⁷ JSON - JavaScript Object Notation - tekstualno baziran standardni format, dizajniran za razmenu poruka razumljivih ljudima.

- Oporavak u datom trenutku (*engl. point-in-time recovery*) - Funkcionalnost pomaže pri čuvanju podataka u slučaju slučajnih operacija čitanja/upisa i kontinuirano pravljenje rezervnih kopija podataka DynamoDB tabela koje su dostupne za oporavak u narednih 35 dana;
- Enkripcija podataka u mirovaju (*engl. encryption at rest*) - Podaci se šifriraju čak i kada tabele nisu u upotrebi što povećava sigurnost podataka. Podrazumeva se šifriranje pomoću matičnih AWS ključeva.

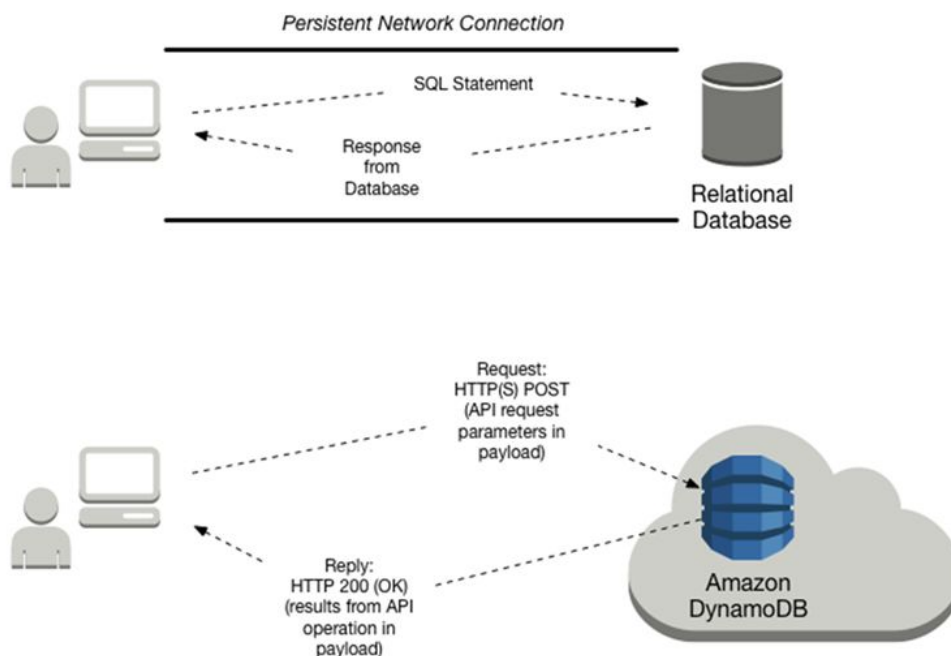
Programski jezici i programski okviri (*engl. framework*) koji se mogu povezati sa DynamoDB bazom su Java, JavaScript, Node.js, Go, C# .NET, Perl, PHP, Python, Ruby, Haskell, Erlang, Django i Grails.

Neke od lošijih strana koje upotreba DynamoDB baze nosi sa sobom su sledeće:

- Komercijalno je rešenje - DynamoDB je u vlasništvu AWS-a i nije otvorenog koda za sada, tako da je prebacivanje na drugi servis potencijalno skup i zahtevan proces;
- Nema ugrađenog keširanja - DynamoDB zahteva da pristup podacima bude uglavnom uniforman, odnosno ukoliko se pristupa određenom segmentu podataka (na primer najskorijim unosima) više nego drugim segmentima, keširanje najskorijih stavki neće biti moguće. U tom slučaju se koristi *DynamoDB Accelerator (skr. DAX)* uz dodatne troškove;
- Ne pruža podršku za JOIN operacije - DynamoDB dizajn ne dozvoljava pridruživanje podataka iz više tabela. Klasičan način implementiranja operacije JOIN može biti skuplji, sporiji i složeniji u odnosu na relacione baze.

Kod relacionih baza klijentska aplikacija ostvaruje i održava konekciju sa bazom, sa druge strane, DynamoDB predstavlja web uslugu i interakcije sa njim nemaju predodređeno stanje (*engl. stateless*). Aplikacije koje koriste DynamoDB ne moraju da održavaju postojeane mrežne veze, već se interakcija realizuje korišćenjem HTTP(S) zahteva i odgovora.

Aplikacije koje se povezuju na neku relacionu bazu izdaju SQL izraze (*engl. statement*) za svaku operaciju koju žele da izvrše, a RDBMS nakon prijema proverava sintaksu, kreira plan za izvođenje operacije, a zatim izvršava operaciju prema tom planu. RDBMS vraća rezultate SQL izraza ili status i poruku greške ukoliko iz nekog razloga nisu mogli da se izvrše. Aplikacije koje koriste DynamoDB šalju HTTP(S) zahteve bazi koji sadrže naziv operacije i odgovarajuće parametre, a po prijemu zahteva on se odmah izvršava u DynamoDB okruženju. DynamoDB vraća HTTP(S) odgovor koji sadrži rezultate operacije ili status HTTP greške i poruku.



Slika2. Poređenje komunikacije klijenta sa relacionim bazama i Amazon DynamoDB

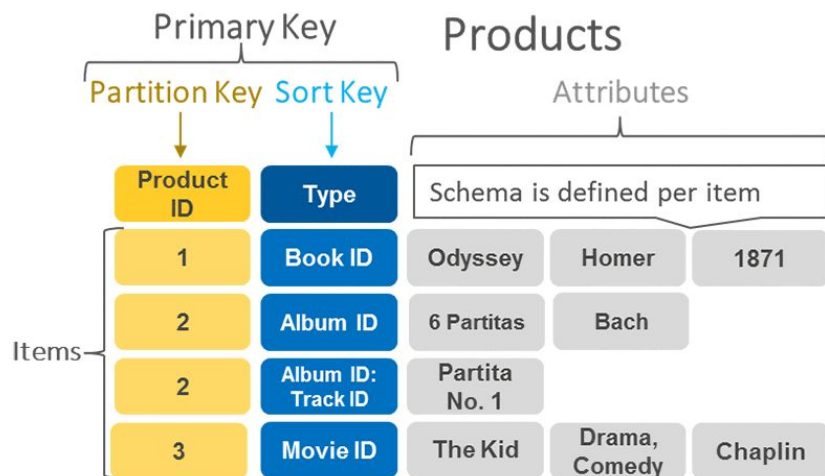
Tipovi podataka koji su podržani u DynamoDB su:

- Skalari - *Number, String, Binary, Boolean* i *Null*;
- Viševrednostni (*engl. multi-valued*) - *String Set, Number Set* i *Binary Set*
Ovi tipovi podataka su svi setovi, što znači da vrednosti elemenata moraju biti jedinstveni.
- Dokumenti - *List* i *Map*.

DynamoDB koristi JSON kao sintaksni jezik, a naredba za pravljenje tabele zahteva samo tri argumenta: naziv tabele (*engl. TableName*), primarni ključ (*engl. KeySchema*) i listu atributa (*engl. AttributeDefinitions*).

Osnovne komponente DynamoDB baze

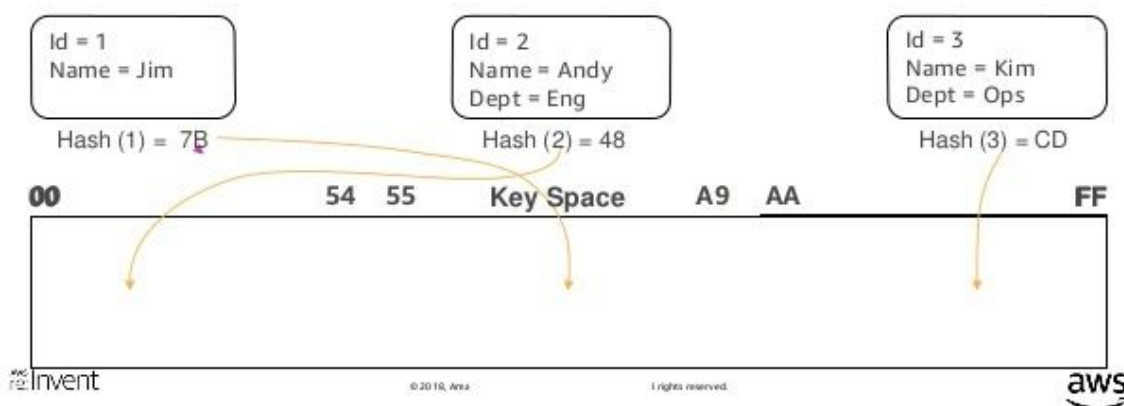
DynamoDB sastoji se od **tri osnovne jedinice - tabela, atribut i stavke**. Tabela sadrži skup stavki, stavka sadrži skup atributa, a atribut je najjednostavniji element koji sadrži podatke. Tabele imaju sličan koncept kao kod drugih sistema za upravljanje bazama podataka. Svaka tabela sastoji se od 0 ili više stavki (*engl. items*). Stavke su ekvivalent torki odnosno jednom redu u tabeli kod relacionih baza podataka. Kod DynamoDB baze ne postoji limit za broj stavki koje može da sadrži jedna tabela. Atributi su ekvivalent kolonama kod relacionih i sličnih baza.



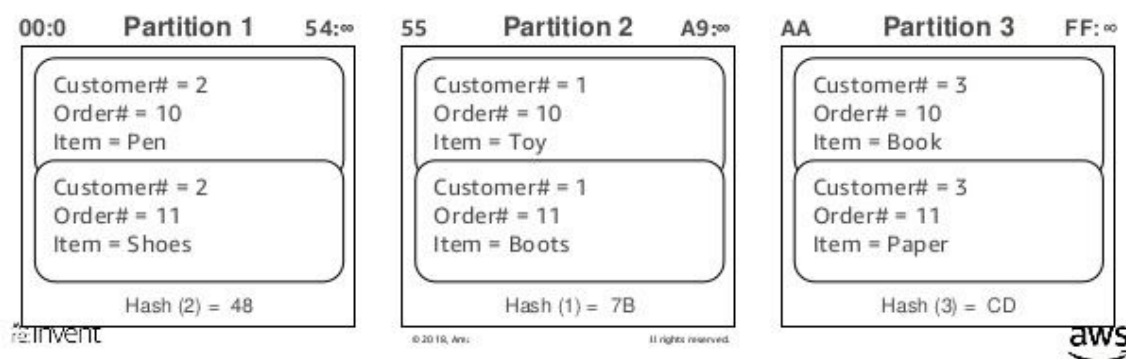
Slika3. Prikaz primera tabele "Proizvodi"

DynamoDB zahteva postojanje primarnog ključa kao atributa koji mora biti jedinstven, tipa *string*, *number* ili *binary* i podržava dve vrste primarnih ključeva (skr. PK):

1. **Ključ particionisanja** (*engl. partition key ili hash type PK*) - Atribut koji jedinstveno identifikuje stavku gradi se uz pomoć hash indeksa. On je obavezan u DynamoDB tabeli. Ukoliko postoji samo on PK je jednovrednosni ključ i mora biti jedinstven, a on određuje particiju odnosno fizički server na kome je smeštena određena stavka. [Slika4]
2. **Ključ sortiranja** (*engl. sort key ili range type PK*) - Ova vrsta PK izgrađena je na heširanom ključu i ključu raspona u tabeli. Ključ sortiranja je opcioni deo PK, koji uz ključ particionisanja čini kompozitni PK. [Slika5] Uz pomoć ključa za sortiranje određuje se redosled u kome su stavke smeštene na određenoj particiji. Ukoliko postoji kompozitni PK ključ particionisanja ne mora biti jedinstven, ali kombinacija ključa particionisanja i sortiranja mora jedinstveno određivati jednu stavku. Sve stavke sa istim ključem particionisanja se smeštaju na istoj fizičkoj lokaciji.

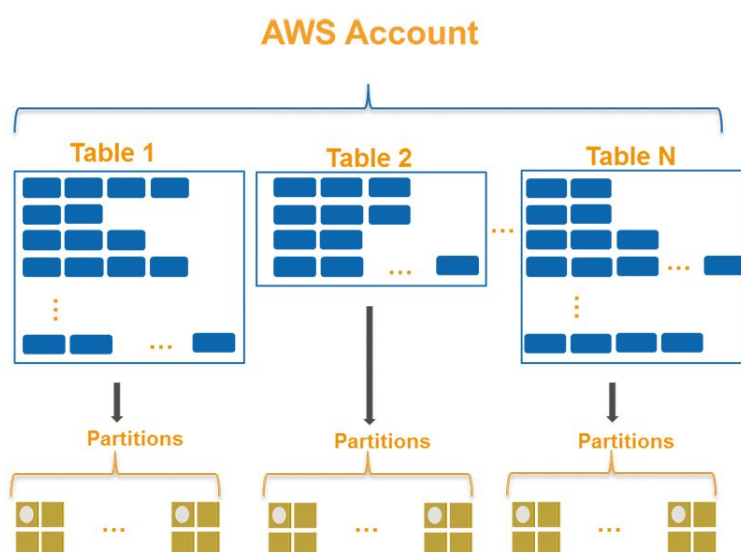


Slika4. Primer za ključ particionisanja (hash ključ)



Slika5. Primer za kompozitni PK koji čine ključ partitionisanja i ključ sortiranja

Partitionisanje podataka



Slika6. Ilustracija podele tabela po fizičkim particijama u DynamoDB bazi

Particija predstavlja alokirani prostor za tabelu koja se smešta na SSD memorije (*engl. solid state drive*) i automatski se replicira kroz više zona dostupnosti (*engl. Availability Zones*) u AWS regionu. Partitionisanje vrši DynamoDB sistem bez posebne podrške ili potrebe za podešavanjem.

Podaci u DynamoDB bazi su partitionisani automatski korišćenjem hash ključa (ključ partitionisanja), a logika raspodele po particijama zavisi od veličine tabelle i propusnosti.

$$BP_p = \frac{RCU}{3000 RCU} + \frac{WCU}{1000 WCU}$$

$$BP_v = \frac{Veličina\ tabelle\ [GB]}{10\ GB}$$

$$BP = \max(BP_p, BP_v)$$

BP_p - broj particija na osnovu propusnosti

BP_v - broj particija na osnovu veličine tabelle

RCU (*engl. Read Capacity Units*) - jedinica kapaciteta za operaciju čitanja, meri se na svakih 4KB/s

WCU (*engl. Write Capacity Units*) - jedinica kapaciteta za operaciju upisa, meri se na svakih 1KB/s

Podrazumevano podešavanje particije je da može da sadrži do 10GB podataka, a da izdrži do 1000 WCU i 3000 RCU. Jedna RCU predstavlja jednu operaciju jako konzistentnog (*engl. strongly consistent*) čitanja u sekundi ili dve operacije eventualno konzistentnog (*engl. eventually consistent*) čitanja u sekundi.

Pri pravljenju tabele, njen inicijalni status je **CREATING** i tokom trajanja ove faze DynamoDB alocira dovoljno particija za tabelu kako bi mogla da zadovolji uslove propusnosti. Kada tabela pređe u status **ACTIVE** moguće je vršiti operacije upisa i čitanja podataka.

Primer1: Za slučaj u kome imamo tabelu veličine 16 GB i vrednosti od 6000 za RCU i 1000 za WCU dobijamo sledeće:

$$BP_p = \frac{6000 RCU}{3000 RCU} + \frac{1000 WCU}{1000 WCU} = 2 + 1 = 3$$

$$BP_v = \frac{16 GB}{10 GB} = 1.6 \approx 2$$

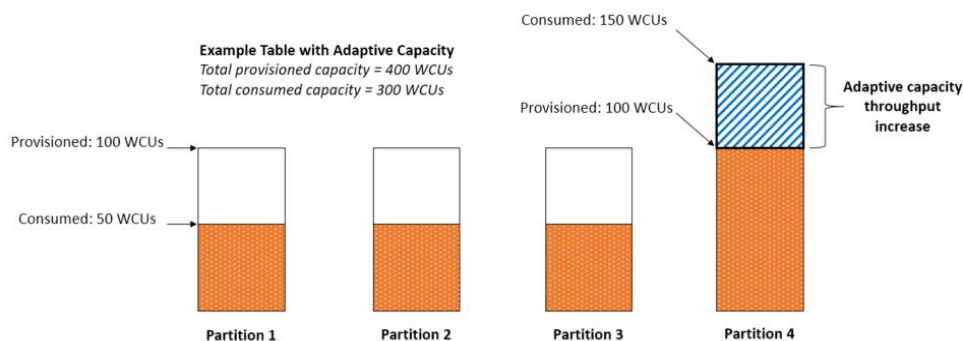
$$BP = \max(3, 2) = 3$$

Što znači da će se podaci smestiti na 3 particije. Raspodela RCU i WCU će biti uniformna, odnosno 3000/3 za RCU i 1000/3 za WCU, a sa strane podataka 16/3 = 5.4GB će biti zauzeto po particiji.

DynamoDB alocira dodatne particije za smeštanje tabele u sledećim situacijama:

- Ukoliko se povećaju podešavanja propusnosti preko granice koju mogu da zadovolje trenutne particije;
- Ukoliko postojeća particija bude popunjena do maksimalnog kapaciteta, a potrebno je dodatnog prostora.

Na Slika7 prikazano je kako funkcioniše **adaptivna kapacitivnost**. Primer se osniva na tabeli sa 400 jedinica kapaciteta za operacije upisa (skr. WCU) koje su podjednako raspoređene kroz 4 particije, što omogućava održivost od maksimalno 100 WCU u sekundi. Particije 1, 2 i 3 svaka dobijaju po 50 WCU u sekundi, a particija 4 150 WCU u sekundi. Particija 4 ne može da izdrži toliku količinu saobraćaja, ali zbog adaptivne kapacitivnosti DynamoDB poveća njen kapacitet automatski.



Slika7. Ilustracija adaptivne kapacitivnosti sa 4 particije u DynamoDB bazi

Upravljanje particijama je automatizovan pozadinski proces čiji je rezultat transparentan korisničkoj aplikaciji. Tabele ostaju dostupne za operacije čitanja i/ili upisa tokom celog procesa upravljanja particijama.

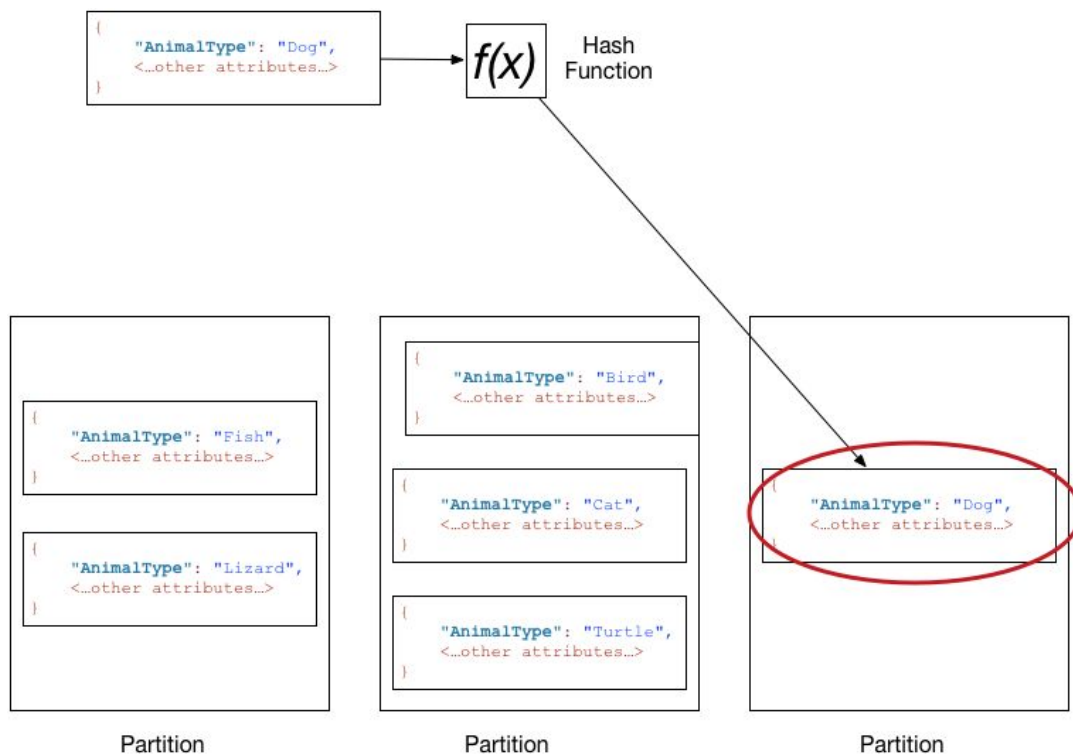
Kako bi se smanjila šansa za gubitak podataka, DynamoDB koristi dvoslojni sistem za oporavak (*engl. backup*) koji se sastoji iz replikacije baze podataka i dugotrajnog skladišta. Svaka particija replicira svoj sadržaj na 3 čvora koji sadrže njenu kopiju. Svaki čvor sadrži dve strukture: B-stablo (*engl. B-tree*) za lociranje stavki i zapis replikacije (*engl. replication log*) u kome su zabeležene sve promene na tom čvoru. Periodično se dogodi snapshot ove dve strukture i čuva na S3 mesec dana kako bi *point-in-time recovery* bio omogućen bez problema. Jedan od 3 čvora replikacije postavljen je za vođu (*engl. leader node*), tako da sve operacije upisa idu kroz njega i propagiraju se ostalim čvorovima, što čini operacije upisa konzistentnim. Kako bi vođa-čvor održao svoj status potrebno je da šalje signal svakom čvoru na svakih 1.5 sekundi (*engl. heartbeat*). Za slučaj da vođa-čvor prestane da šalje signal pokreću se izbori za novog vođu za šta se koristi Paxos algoritam⁸.

AutoAdmin se pokreće kada neki od čvorova prestane da odgovara ili kopira podatke sa drugog čvora. Kada particija pređe bilo koju od svojih podrazumevanih maksimalnih vrednosti (RCU, WCU, veličinu skladišta) AutoAdmin će automatski dodati nove particije kako bi se dalje segmentirali podaci. AutoAdmin ima ulogu administratora DynamoDB baze.

DynamoDB koristi **ključ particionisanja** kao ulaznu vrednost hash funkcije koja određuje na kojoj particiji će biti smeštena stavka koja se upisuje u tabelu. Takođe, kada se vrši čitanje određene stavke navodi se njen ključ particionisanja kao ulazni podatak hash funkcije koja vraća referencu particije na kojoj se nalazi tražena stavka.

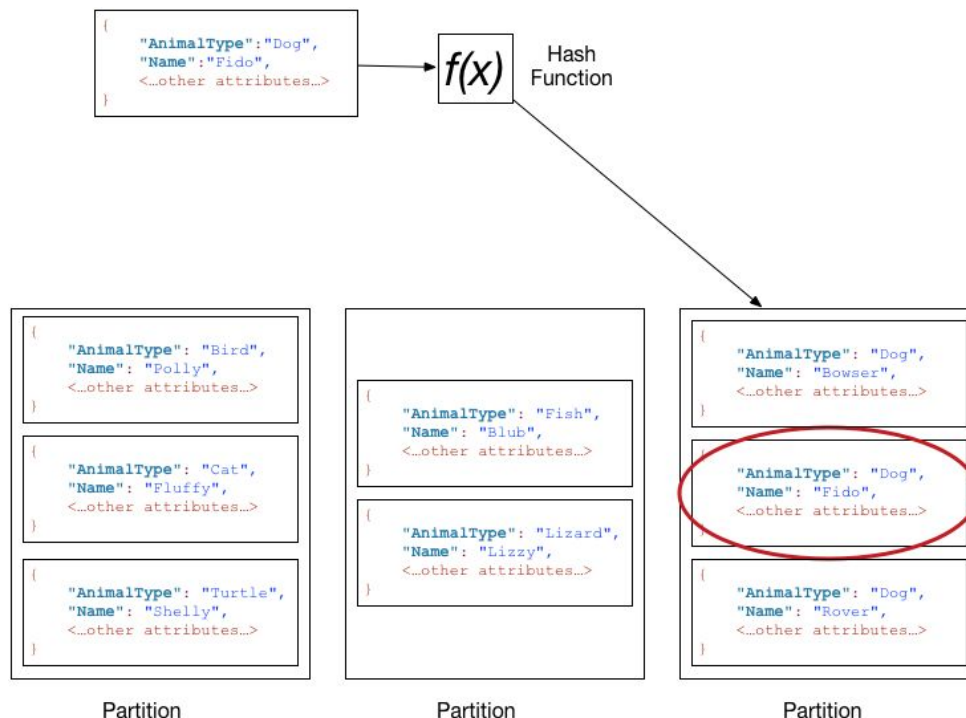
Na Slika8 može se primetiti da je tabela *Pets* podeljena na više particija i ima atribut *AnimalType* kao PK. Primer koji ilustruje na kojoj će particiji biti smeštena nova stavka sa navedenim atributom *AnimalType* = "Dog" dat je na Slika8. Sve stavke sa istim PK smeštene su na istoj particiji.

⁸ *Paxos algoritmi* - predstavljaju familiju protokola za rešavanje koncenzusa u mrežama sa nepouzdanim procesorima.



Slika8. Primer smeštanja nove stavke na određenu particiju sa ključem particionisanja

Slika9 prikazuje primer smeštanja nove stavke na određenu particiju ukoliko je PK **kompozitni ključ**. Proces određivanja particije je isti kao i u prethodnom slučaju, odnosno ona se određuje samo na osnovu vrednosti ključa particionisanja, ali se uz pomoć ključa sotriranja ovoga puta stavka smešta na uređen način. U ovom slučaju PK se sastoji iz ključa particionisanja *AnimalType* i ključa sotriranja *Name*.



Slika9. Primer smeštanja nove stavke na određenu particiju sa kompozitnim PK

DynamoDB indeksi

U DynamoDB bazi ne postoji optimizator upita (*engl. query optimizer*) i indeksi su jednostavno odvojene tabele sa različitim ključevima koje referenciraju originalnu tabelu nad kojom su kreirani. Kada se kreira DynamoDB indeks pravi se i nova kopija podataka, ali samo za polja koja su navedena odnosno barem ona koja se indeksiraju i originalni PK.

Postoje dva tipa indeksa u DynamoDB bazi, i to:

1. **Lokalni sekundarni indeks (*engl. Local Secondary Index*)** - Kod ovog tipa indeksa neophodno je imati kompozitni PK i to ključ partitionisanja koji je isti kao u originalnoj tabeli i ključ sortiranja koji se mora razlikovati od indeksirane tabele.
2. **Globalni sekundarni indeks (*engl. Global Secondary Index*)** - Za ovaj tip indeksa nije neophodno imati kompozitni PK. U ovom slučaju ključ partitionisanja i ključ sortiranja mogu biti različiti od originalne tabele koja se indeksira.

Indeksiranje nije obavezno, ali omogućava veću fleksibilnost pri izvršavanju upita. Svaka tabela u DynamoDB bazi ima ograničenje od 20 globalnih sekundarnih indeksa i 5 lokalnih sekundarnih indeksa. Globalni sekundarni indeksi su raspoređeni po particijama, kao i originalne tabele, ali se čuvaju odvojeno od originalnih podataka. Sekundarni indeksi ne bi trebalo da se koriste za attribute prema kojima se ne pretražuje često, zbog zauzimanja dodatnog prostora i zadržavanja na optimalnijem stanju baze.

The screenshot shows the 'Add index' dialog in the AWS Management Console. The dialog is titled 'Add index' and has a close button (X). It contains the following fields and options:

- Primary key***: Partition key. The selected key is 'PostedBy' with a type of 'String'.
- Add sort key**: A checked checkbox.
- Sort key**: The selected key is 'Message' with a type of 'String'.
- Index name***: The selected name is 'PostedBy-Message-Index'.
- Projected attributes**: A dropdown menu set to 'All'.
- Create as Local Secondary Index**: An unchecked checkbox.
- Buttons**: 'Cancel' and 'Add index'.

The background shows the 'Create DynamoDB table' page with a table named 'Table n' and a primary key 'Primary'.

Slika10. Primer kreiranja globalnog sekundarnog indeksa "PostedBy-Message-Index"

DynamoDB Streams

DynamoDB tokovi podataka (*engl. streams*) predstavljaju opcionu mogućnost koja pamti sve modifikacije izvršene nad podacima u DynamoDB tabelama u skoro realnom vremenu i u redosledu u kome su se one dogodile. DynamoDB Stream predstavlja servis za praćenje toka podataka koji se upisuje u tabelu ili čita iz tabele. Svaki događaj koji je izazvao neku promenu nad podacima predstavljen je **zapisom toka podataka** (*engl. stream record*). Zapis toka podataka se ispiše u slučaju da je omogućen tok podataka nad tabelom i kada se dogodi neki od sledećih događaja koji uzrokuje modifikaciju podataka:

- Nova stavka je dodata u tabelu - U zapisu se pamti kopija stavke sa svim njenim atributima;
- Stavka je ažurirana - U zapisu se pamti kopija pre i kopija stavke posle ažuriranja za one attribute koji su promenjeni;
- Stavka je obrisana iz tabele - Zapis pamti kopiju stavke u stanju u kojem je bila pre njenog brisanja.

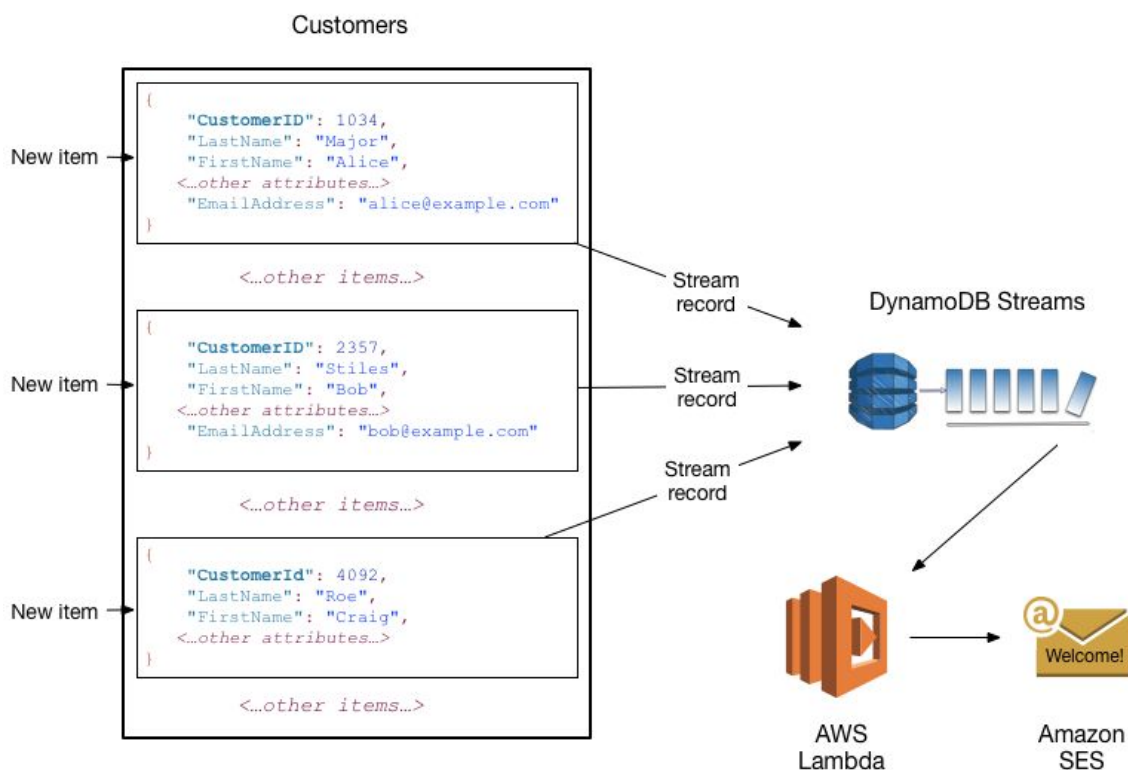
Svaki zapis toka podataka sadrži i naziv tabele, vremensku oznaku (*engl. timestamp*) i druge metapodatke. Zapisi toka podataka imaju životni vek od 24h, nakon čega se automatski uklanjaju iz toka podataka. Komande koje se koriste uz DynamoDB tokove podataka su:

- *ListStream* - vraća listu svih tokova podataka;
- *DescribeStream* - vraća detaljni opis o toku podatka i resursima koje koristi;
- *GetShardIterator* - vraća posebnu strukturu podataka - *Shard Iterator*, koja čuva informacije o toku podataka;
- *GetRecords* - vraća informacije o toku uz pomoć *Shard Iterator* strukture.

Tokovi podataka se mogu koristiti uz AWS Lambda⁹ za kreiranje trigger-a koji će automatski biti izvršeni čim se dogodi određena promena u toku podataka. Slika11 ilustruje primer za tabelu *Customers* sa tokom podataka i upotrebom Lambda funkcije za slanje elektronske pošte svim novim mušterijama iz tabele. Svaki put kada bi se nova stavka pojavila u tabeli Lambda funkcija bi bila izvršena ukoliko postoji atribut koji joj je potreban za izvršenje (na primer *EmailAddress*) uz pomoć Amazon Simple Email servisa¹⁰ (*skr. Amazon SES*).

⁹ *AWS Lambda* - predstavlja servis koji pokreće neki kod kao odgovor na određeni događaj i automatski upravlja resursima na koje on utiče.

¹⁰ *Amazon SES* (*engl. Amazon Simple Email Service*) - predstavlja cloud servis za slanje elektronske pošte dizajniran da pomogne pri kontaktiranju korisnika, obavljanja transakcija i slanja drugih obaveštenja.



Slika 11. Primer kombinovanja DynamoDB Stream-a sa AWS Lambda funkcijom i Amazon SE servisom

Konzistentnost operacija Read/Write

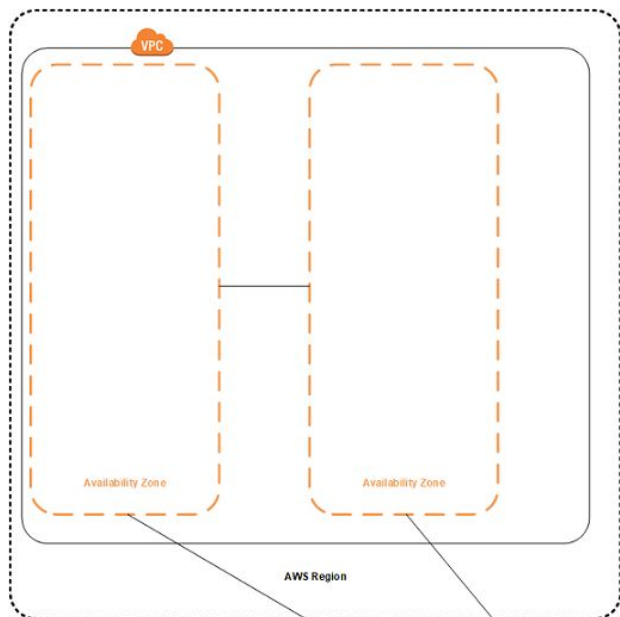
Amazon DynamoDB je dostupan u više izolovanih i nezavisnih **AWS regiona** na svetu. Moguće je imati tabele sa istim nazivom u okviru različitih AWS regiona, jer se one smatraju potpuno odvojenim tabelama. Svaki AWS region sastoji se iz više različitih lokacija koje predstavljaju **zone dostupnosti** (*engl. Availability Zones*). Svaka zona dostupnosti je izolovana od pada u drugim zonama dostupnosti i pruža mrežnu konekciju sa drugim zonama dostupnosti u okviru istog AWS regiona. Ova mogućnost dozvoljava brzu replikaciju podataka između više zona dostupnosti u AWS regionu. **Lokalne zone** (*engl. Local Zones*) predstavljaju produžetak AWS regiona koje su geografski bliže korisnicima koji im pristupaju.

DynamoDB omogućava **eventualno konzistentna čitanja i jako konzistentna čitanja**.

Čitanje iz DynamoDB tabele koje je jako konzistentno sa sobom donosi i neke poteškoće kao što su:

- U slučaju mrežnih problema DynamoDB će vratiti serversku grešku HTTP 500;
- Postoji visoka latentnost;
- Ne podržavaju se nad globalnim sekundarnim indeksima.

Operacije *Put*, *Update* ili *Delete*, odnosno neka od **operacija upisa** šalje se kroz zahtev i nakon autentikacije i autorizacije preračunava se ključ particionisanja hash funkcijom i upisuje na čvor vođu (*engl. leader node*), a zatim i na jednu repliku. Nakon toga se vraća



poruka o uspešnom upisu i propagira upis na treći čvor replikacije. Ovaj način obezbeđuje **konzistentnost kod operacija upisa**.



Slika12. Šematski prikaz arhitekture AWS regiona, zona dostupnosti i lokalnih zona

Pristup DynamoDB

Pristup DynamoDB bazi je jednostavan i omogućuje se uz pomoć sledećih metoda:

- **Konzola (engl. Console)** - Amazon web servis;
- **CLI (engl. Command Line Interface)** - Putem *Command Prompt* prozora sa računara;
- **Upotrebom API-ja** - sa podrškom za mnoge programske jezike i okruženja kao što su Java, JavaScript, .NET, Python, PHP i drugi.

Slika13 ilustruje pravljenje nove tabele *Reply* upotrebom konzole na AWS sajtu. Tabela ima kompozitni PK i čine ga *Id* kao ključ particionisanja i *ReplyDateTime* kao ključ sortiranja.

Create DynamoDB table

Tutorial ?

DynamoDB is a schema-less database that only requires a table name and primary key. The table's primary key is made up of one or two attributes that uniquely identify items, partition the data, and sort data within each partition.

Table name* ⓘ

Primary key* Partition key

ⓘ

☒ Add sort key

ⓘ

Table settings

Default settings provide the fastest way to get started with your table. You can modify these default settings now or after your table has been created.

☐ Use default settings

Slika13. Prikaz pravljenja tabele Reply preko konzole

Na Slika14 vidi se pristup AWS-u putem *Command Prompt*-a sa računara. Prikazane su i dve komande, prva kojom se upisuju stavke iz JSON fajla u tabelu *Reply* koja je prethodno napravljena u DynamoDB bazi, a druga koja vraća sve tabele koje postoje u bazi.

```
Command Prompt

"TableNames": [
  "Forum",
  "ProductCatalog",
  "Reply",
  "Thread"
]
}

C:\Users\Lenovo\Desktop\sampladata>aws configure get region
us-east-2

C:\Users\Lenovo\Desktop\sampladata>aws dynamodb batch-write-item --request-items file://Reply.json
{
  "UnprocessedItems": {}
}

C:\Users\Lenovo\Desktop\sampladata>aws dynamodb list-tables
{
  "TableNames": [
    "Forum",
    "ProductCatalog",
    "Reply",
    "Thread"
  ]
}
```

Slika14. Pristup DynamoDB bazi iz Command Prompt prozora

Pristup preko API-ja definiše se kroz 4 oblasti:

- kontrolna oblast,
- oblast podataka,
- tokovi podataka
- i transakcije.

Kontrolna oblast (engl. *Control plane*) pruža mogućnost pravljenja i upravljanja DynamoDB tabelama uz opcije:

- *CreateTable* - Pravljenje nove tabele, uz opciono pravljenje sekundarnih indeksa i omogućavanje tokova podataka za datu tabelu;
- *DescribeTable* - Vraća informacije o tabeli, kao što su PK, indeksi i slično;

- *ListTables* - Vraća niz imena svih tabela;
- *UpdateTable* - Ažurira podešavanja tabele ili indeksa tabele, pravi nove ili modifikuje postojeće indekse i tokove podataka;
- *DeleteTable* - Briše tabelu i sve njene zavisne objekte iz DynamoDB baze.

Oblast podataka (engl. *Data plane*) omogućava korišćenje CRUD operacija nad podacima u tabelama:

- Kreiranje podataka
 - *PutItem* - Upisuje jednu stavku u tabelu. Moraju se navesti atributi koji čine PK;
 - *BatchWriteItem* - Omogućava upis do 25 stavki u tabelu. Može se koristiti i kod brisanja [navedeno pod stavkom Brisanje podataka u nastavku];
- Čitanje podataka
 - *GetItem* - Vraća jednu stavku iz tabele. Mora se navesti PK, mogu se vratiti svi atributi ili određeni skup atributa;
 - *BatchGetItem* - Vraća do 100 stavki iz jedne ili više tabela;
 - *Query* - Vraća sve stavke sa navedenim ključem particionisanja; Ukoliko postoji i ključ sortiranja može se koristiti za sužavanje izbora stavki koje je potrebno prikazati. Može se koristiti i nad indeksima;
 - *Scan* - Vraća sve stavke iz navedene tabele ili sa navedenim indeksom. Mogu se vratiti svi atributi ili samo njihov podskup, a može se i dalje suziti izbor postavljanjem uslova;
- Ažuriranje podataka
 - *UpdateItem* - Ažurira jedan ili više atributa određene stavke. Mora se navesti PK stavke koju je potrebno ažurirati. Mogu se dodati novi ili izmeniti ili obrisati postojeći atributi i izvesti razna uslovljena ažuriranja;
- Brisanje podataka
 - *DeleteItem* - Brisanje jedne stavke iz tabele sa obaveznom navođenjem PK;
 - *BatchWriteItem* - Brisanje do 25 stavki iz jedne ili više tabela.

Oblast tokova podataka (engl. *streams*) omogućava podešavanje toka podataka nad tabelom uz pomoć:

- *ListStreams* - Vraća niz svih tokova podataka za konkretnu tabelu;
- *DescribeStream* - Vraća informacije o toku podataka, kao što je njegov *Amazon Resource Name* (skr. ARN) i gde aplikacija može da započne čitanje prvih nekoliko zapisa toka podataka;
- *GetShardIterator* - Vraća *Shard Iterator*, posebnu strukturu podataka koja se koristi za vraćanje zapisa toka podataka;
- *GetRecords* - Vraća jedan ili više zapis toka podataka uz dati *Shard Iterator*.

Oblast transakcija (engl. *transactions*) pruža atomičnost, konzistentnost i trajnost odnosno ACID svojstva i omogućava održavanje korektnosti podataka unutar aplikacije, uz komande:

- *TransactWriteItems* - Grupna operacija koja omogućava Put, Update i Delete operacije nad više stavki unutar jedne ili više tabela koja se izvršava ili u potpunosti ili se roll-back-uje na prethodno stanje;
- *TransactGetItems* - Grupna operacija koja omogućava Get za vraćanje više stavki iz jedne ili više tabela.

U nastavku je prikazan primer upotrebe API-ja za jezik Python na bazi koja treba da čuva informacije o knjigama za potrebe jedne knjižare i ima sledeću šemu:

Naziv Atributa	Tip	Opis
Title	String	Naslov knjige
Author	String	Ime i prezime autora knjige
Category	String	Kategorija kojoj knjiga pripada, na primer Biografija, Drama, Naučna fantastika i slično
Formats	Map	Svi formati u kojima je knjiga dostupna, na primer audio knjiga, tvrdi povez i slično; koji su mapirani na brojeve u inventaru

Tabela2. Primer šeme DynamoDB table za čuvanje informacija o knjigama

Pokretanjem komande `python create_table.py` pokreće se skripta koja pravi tabelu na osnovu šeme iz Tabele2.

Ova skripta sadrži u sebi sledeći kod:

```
import boto3
client = boto3.client('dynamodb', region_name='us-east-2')
try:
    resp = client.create_table(
        TableName="Books",
        KeySchema=[
            {
                "AttributeName": "Author",
                "KeyType": "HASH"
            },
            {
                "AttributeName": "Title",
                "KeyType": "RANGE"
            }
        ],
        AttributeDefinitions=[
            {
                "AttributeName": "Author",
                "AttributeType": "S"
            },
            {
                "AttributeName": "Title",
                "AttributeType": "S"
            }
        ]
    )
```

```

    }},
    ProvisionedThroughput={
        "ReadCapacityUnits": 1,
        "WriteCapacityUnits": 1
    })
print("Table created successfully!")
except Exception as e:
    print("Error creating table:")
    print(e)

```

Dalje uz komandu `python insert_items.py` u tabelu se dodaju stavke, a skripta ima sledeći sadržaj:

```

import boto3
dynamodb = boto3.resource('dynamodb', region_name='us-east-2')
table = dynamodb.Table('Books')
with table.batch_writer() as batch:
    batch.put_item(Item={"Author": "John Grisham", "Title": "The Rainmaker",
        "Category": "Suspense", "Formats": { "Hardcover": "J4SUKVGU", "Paperback":
"D7YF4FCX" } })
    batch.put_item(Item={"Author": "John Grisham", "Title": "The Firm",
        "Category": "Suspense", "Formats": { "Hardcover": "Q7QWE3U2",
        "Paperback": "ZVZAYY4F", "Audiobook": "DJ9KS9NM" } })
    batch.put_item(Item={"Author": "James Patterson", "Title": "Along Came a Spider",
        "Category": "Suspense", "Formats": { "Hardcover": "C9NR6RJ7",
        "Paperback": "37JVGZG", "Audiobook": "6348WX3U" } })
    batch.put_item(Item={"Author": "Dr. Seuss", "Title": "Green Eggs and Ham",
        "Category": "Children", "Formats": { "Hardcover": "GVJZQ7JK",
        "Paperback": "A4TFUR98", "Audiobook": "XWMGHW96" } })
    batch.put_item(Item={"Author": "William Shakespeare", "Title": "Hamlet",
        "Category": "Drama", "Formats": { "Hardcover": "GVJZQ7JK",
        "Paperback": "A4TFUR98", "Audiobook": "XWMGHW96" } })

```

Dodavanje globalnog sekundarnog indeksa pokretanjem skripte `add_secondary_index.py` dodaje se indeks nad atributom *Category*, koji će omogućiti da se vrate sve knjige za traženu kategoriju. Kada se dodaje globalni sekundarni indeks za postojeću tabelu, DynamoDB asinhrono popunjava tabelu indeksa sa postojećim stavkama u tabeli i moguće je koristiti indeks tek nakon što se sve stavke dodaju odnosno status indeksa postane **ACTIVE**. Skripta sadrži sledeći kod:

```

import boto3
client = boto3.client('dynamodb', region_name='us-east-2')
try:
    resp = client.update_table(
        TableName="Books",
        AttributeDefinitions=[
            {
                "AttributeName": "Category",
                "AttributeType": "S"
            },
        ],
        GlobalSecondaryIndexUpdates=[
            { "Create": {
                "IndexName": "CategoryIndex",
                "KeySchema": [
                    {
                        "AttributeName": "Category",
                        "KeyType": "HASH"
                    },
                ],
                "Projection": {

```

```

        "ProjectionType": "ALL"
    },
    "ProvisionedThroughput": {
        "ReadCapacityUnits": 1,
        "WriteCapacityUnits": 1,
    }},)
print("Secondary index added!")
except Exception as e:
    print("Error updating table:")
    print(e)

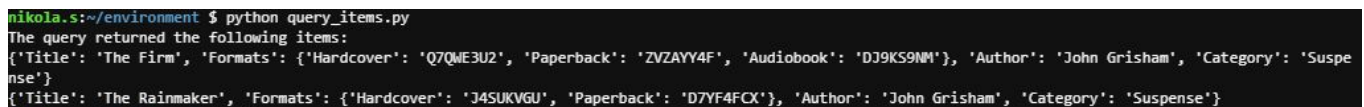
```

Ukoliko želimo da prikazemo sve knjige određenog autora šaljemo upit nad tabelom uz pomoć skripte [query_items.py](#), koja sadrži Python kod u nastavku:

```

import boto3
from boto3.dynamodb.conditions import Key
dynamodb = boto3.resource('dynamodb', region_name='us-east-2')
table = dynamodb.Table('Books')
resp = table.query(KeyConditionExpression=Key('Author').eq('John
Grisham'))
print("The query returned the following items:")
for item in resp['Items']:
    print(item)

```



```

nikola.s:~/environment $ python query_items.py
The query returned the following items:
{'Title': 'The Firm', 'Formats': {'Hardcover': 'Q7QME3U2', 'Paperback': 'ZVZAYY4F', 'Audiobook': 'DJ9KS9NM'}, 'Author': 'John Grisham', 'Category': 'Suspe
nse'}
{'Title': 'The Rainmaker', 'Formats': {'Hardcover': 'J4SUKVGU', 'Paperback': 'D7YF4FCX'}, 'Author': 'John Grisham', 'Category': 'Suspense'}

```

Slika15. Prikaz rezultata upita nad tabelom Books za navedenog autora

Slanje upita nad indeksiranom tabelom uz prethodno kreiran indeks nad atributom *Category* vraća sve knjige sa navedenom kategorijom preko skripte [query_with_index.py](#), koja sadrži sledeći kod:

```

import time
import boto3
from boto3.dynamodb.conditions import Key
dynamodb = boto3.resource('dynamodb', region_name='us-east-2')
table = dynamodb.Table('Books')
while True:
    if not table.global_secondary_indexes or
table.global_secondary_indexes[0]['IndexStatus'] != 'ACTIVE':
        print('Waiting for index to backfill...')
        time.sleep(5)
        table.reload()
    else:
        break
resp = table.query(
    IndexName="CategoryIndex",
    KeyConditionExpression=Key('Category').eq('Suspense'),)
print("The query returned the following items:")
for item in resp['Items']:
    print(item)

```

```

nikola.s:~/environment $ python query_with_index.py
The query returned the following items:
{'Title': 'The Firm', 'Formats': {'Hardcover': 'Q7QNE3U2', 'Paperback': 'ZVZAYY4F', 'Audiobook': 'DJ9KS9NM'}, 'Author': 'John Grisham', 'Category': 'Suspense'}
{'Title': 'The Rainmaker', 'Formats': {'Audiobook': '8WE3KPTP', 'Hardcover': 'J4SUKVGU', 'Paperback': 'D7YF4FCX'}, 'Author': 'John Grisham', 'Category': 'Suspense'}
{'Title': 'Along Came a Spider', 'Formats': {'Hardcover': 'C9NR6RJ7', 'Paperback': '37JVG0Z6', 'Audiobook': '6348MX3U'}, 'Author': 'James Patterson', 'Category': 'Suspense'}
nikola.s:~/environment $

```

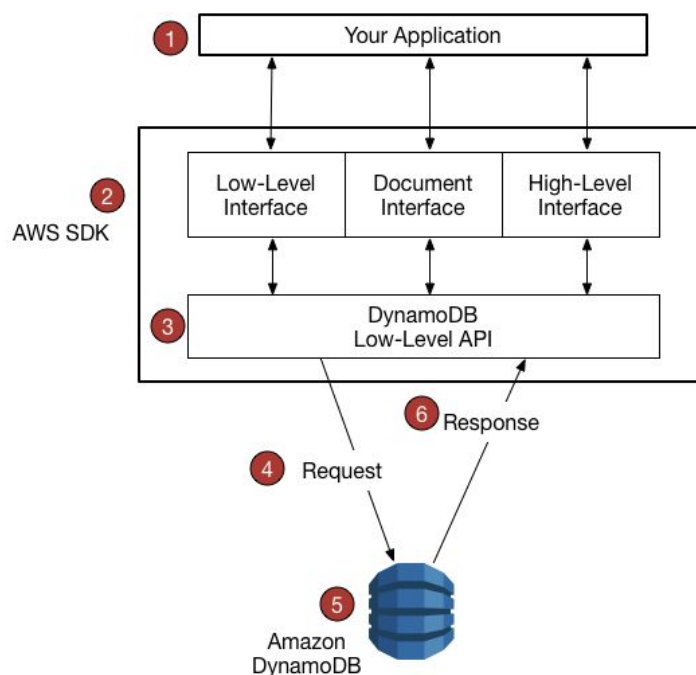
Slika16. Rezultat izvršenja upita nad tabelom Books koja ima kreiran indeks nad atributom Category

AWS SDK podrška za DynamoDB

Ilustracija na Slika17 prikazuje pregled kreiranja aplikacije koja koristi **Amazon DynamoDB uz podršku AWS SDK**¹¹. Korisnička aplikacija se programira uz pomoć odgovarajućeg AWS SDK za korišćeni programski jezik, AWS SDK sam konstruiše HTTP(S) zahteve za DynamoDB API na niskom nivou i prosleđuje ih bazi. Ukoliko je zahtev uspešan DynamoDB odgovara sa standardnim HTTP 200 (OK) kodom, u suprotnom vraća kod greške i poruku koji dalje procesira AWS SDK i propagira nazad do aplikacije.

Svaki AWS SDK bavi se i sledećim zadacima:

- Formatiranje HTTP(S) zahteva i serijalizacija parametara;
- Generisanje kriptografskih potpisa za svaki zahtev;
- Prosleđivanje zahteva DynamoDB bazi i primanje odgovora od nje;
- Obrađivanje rezultata odgovora baze;
- Implementiranje osnovne logike ponovnog pokušaja za slučaj greške.



Slika17. Arhitektura sistema koji se sastoji iz aplikacije, odgovarajućeg AWS SDK i DynamoDB baze

¹¹ AWS SDK (engl. Amazon Web Services Software Development Kit) - skup alata za programiranje koje pruža AWS za određene programske jezike kako bi koristili njihove servise.

Memorijsko ubrzanje - DAX

Amazon DynamoDB je dizajniran za skaliranje i dobre performanse. U većini slučajeva vreme potrebno za odgovor može biti izmereno u milisekundama, a pored toga postoji mogućnost upotrebe **DynamoDB akceleratora** (*engl. **DynamoDB Accelerator**, skr. **DAX***) koji može smanjiti vreme potrebno za odgovor do reda veličine mikrosekundi u određenim slučajevima korišćenja za pristupanje eventualno konzistentim podacima. DAX predstavlja servis za keširanje koji se upotrebljava kod zahtevnijih aplikacija, podržava enkripciju na serverskoj strani i dodatno:

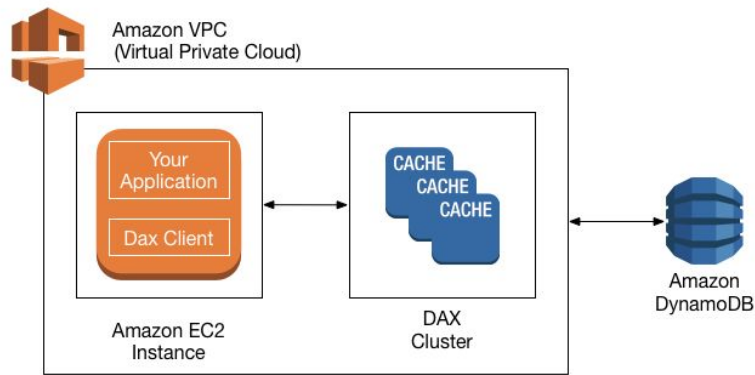
1. Smanjuje vreme potrebno za odgovor za operacije eventualno konzistentnog čitanja sa jednocifrenih milisekundi do mikrosekundi;
2. Smanjuje operativnu i aplikativnu kompleksnost pružanjem servisa koji je kompatibilan sa DynamoDB bazom, što znači da je potrebna minimalna dodatna konfiguracija postojećih projekata za njegovu upotrebu;
3. Pri velikom opterećenju smanjuje RCU i potencijalne operativne troškove. Ovo ima uticaja kod aplikacija koje često vrše čitanje nad individualnim ključevima.

Idealan je za aplikacije kojima je potrebno najbrže vreme za odgovor čitanja kao što su kladenje u realnom vremenu, online berze i slično. Takođe, kod aplikacija koje čitaju mali broj stavki češće nego ostatak baze, kao što je na primer online prodavnica sa akcijom na određeni tip proizvoda. Sa druge strane, DAX nije idealan za aplikacije koje zahtevaju jako konzistentno čitanje, nemaju posebnu potrebu za operacijama čitanja na nivou mikrosekundi ili imaju fokus na upisu podataka više nego čitanju.

DAX je dizajniran da funkcioniše u okviru Amazon Virtual Private Cloud (*skr. **Amazon VPC***) okruženja. Amazon VPC servis definiše virtuelnu mrežu koja liči na tradicionalni data centar uz kontrolu nad opsegom IP adresa, rutiranjem, mrežnim i podešavanjima sigurnosti.

DAX klaster se može napraviti preko AWS konzole i biće pokrenut u okviru podrazumevanog VPC-a, ukoliko se drugačije eksplicitno ne navede. Pokretanjem Amazon EC2 instance i deployment-om aplikacije na njoj zajedno sa DAX klijentom pokreće se sistem ilustrovan na Slika18.

U okviru DAX klastera postoji više čvorova od kojih je jedan primarni čvor, a ostatak klastera čine replike. Tokom rada aplikacije DAX klijent preusmerava sve zahteve koji stižu preko DynamoDB API-ja u DAX klaster koji obrađuje sve zahteve koje može odnosno koji su keširani, dok ostale prosleđuje DynamoDB bazi. Nakon obrade DAX klaster vraća rezultat aplikaciji i kešira ga na svom primarnom čvoru ukoliko se već ne nalazi na njemu. Sve jako konzistentne operacije čitanja DAX prosleđuje direktno DynamoDB i ovi rezultati se ne keširaju u DAX klasteru.



Slika18. Pregled Amazon VPC servisa sa DAX klasterom koji komunicira sa DynamoDB bazom

Stavke u okviru DAX klastera imaju određeno vreme života (*engl. Time To Live, skr. TTL*) koje podrazumevano iznosi 5 minuta. Svakoј stavki koja se kešira dodeljuje se timestamp. Za slučaj da je neka stavka preostala u kešu i nakon isteka svog TTL-a podrazumeva se kao promašaj (*engl. cache miss*) i zahtev se preusmerava na DynamoDB.

DAX predstavlja servis za keširanje koji je posebno dizajniran za pojednostavlјivanje procesa DynamoDB baze, ali je nezavistan servis i potrebno je dodatno ga konfigurisati za rad u određenim slučajevima korišćenja.

Zaključak

Amazon je vodeći provajder cloud servisa i njegovo NoSQL rešenje DynamoDB je predstavnik AWS usluga u vidu *Database-as-a-Service* rešenja. DynamoDB funkcioniše kao Key-Value i Document baza podataka, a zahteva samo primarni ključ i nije potrebno imati definisanu šemu za pravljenje tabele.

Upotrebom DynamoDB baze očekuju se dobre performanse, čak i sa skaliranjem. Može da skladišti bilo koju količinu podataka i podrži bilo koju količinu saobraćaja. Omogućena je i automatska replikacija podataka i postoje dve vrste sekundarnih indeksa.

Sa druge strane, DynamoDB je komercijalno rešenje, koje dozvoljava lako kombinovanje sa drugim AWS servisima bez prevelikog angažovanja, ali potencijalno ograničeno kada su eksterni servisi u pitanju. Keširanje nije omogućeno samostalno uz DyanmoDB rešenje, već uz pomoć drugog AWS servisa pod akronimom DAX i dodatno naplaćivanje. Konkretno prednosti i mane opisane su u okviru ovog rada, kao i njegove karakteristike i način pristupanja DynamoDB rešenju.

Sve u svemu, DynamoDB predstavlja jedno od vodećih cloud DBaaS NoSQL rešenja koje mogu zadovoljiti potrebe različitih kapaciteta i aplikativnih namena. Primenjuje se u mobilnim, web, gejming, Internet of Things i mnogim drugim aplikacijama i korisničkim slučajevima, a njegove jače strane u vidu bezbednosti, automatizacije backup-a i replikacije pružaju mnogim korisnicima pouzdano i sveobuhvatno komercijalno rešenje.

Literatura

- [1] DynamoDB resursi sa zvaničnog Amazon Web Services sajta
<https://aws.amazon.com/dynamodb/resources/>
- [2] C. Patra - "Amazon DynamoDB - Ten Things" - Članak sa sajta CloudAcademy
<https://cloudacademy.com/blog/amazon-dynamodb-ten-things/>
- [3] Wikipedia članak - "Amazon DynamoDB"
https://en.wikipedia.org/wiki/Amazon_DynamoDB
- [4] Wikipedia članak - "Cloud database"
https://en.wikipedia.org/wiki/Cloud_database
- [5] G. Balasubramanian - "Choosing the right DynamoDB Partition Key" - Članak sa zvaničnog Amazon Web Services bloga
<https://aws.amazon.com/blogs/database/choosing-the-right-dynamodb-partition-key/>