Seminarski rad

Predmet: Sistemi za upravljanje bazama podataka

Tema: Sigurnost MongoDB baze podataka

Student: Jelena Đikić, br. ind. 1488

Niš, januar 2024

Sadržaj

1. Uvod....3
2. Zahtevi za obezbeđivanje podataka savremenih aplikacija....4
   2.1. Bezbednosna arhitektura MongoDB skladišta podataka....5
   2.2. Autentifikacija....6
      2.2.1. SCRAM autentifikacija....6
      2.2.2. x.509 autentifikacija sertifikata....8
      2.2.3. LDAP proxy autentifikacija....9
      2.2.4. Kerberos autentifikacija....9
   2.3. Autorizacija....11
      2.3.1. Ugrađene uloge....12
   2.4. Revizija podataka....15
   2.5. Enkripcija podataka....16
      2.5.1. Enkripcija podataka u prenosu (Transport Encryption)....17
      2.5.2. Enkripcija podataka u mirovanju (Encryption at Rest)....18
      2.5.3. Enkripcija podataka u upotrebi (In-use Encryption)....20
3. Praktični deo....23
   3.1. Autentifikacija....23
   3.2. Autorizacija....25
      3.2.1. Kreiranje readOnly korisnika i demonstracija njegovih privilegija....25
      3.2.2. Kreiranje readWrite korisnika i demonstracija njegovih privilegija....26
   3.3. Revizija....27
   3.4. CSFLE Enkripcija....29
      3.4.1. Kreiranje master ključa (CMS)....29
      3.4.2. Kreiranje jedinstvenog indeksa na kolekciji Key Vault....30
      3.4.3. Kreiranje ključa za enkripciju polja....30
      3.4.4. Kreiranje šeme enkripcije....31
      3.4.5. Dodavanje novog dokumenta sa enkrptovanim poljem u bazu....32

4. Zaključak 34
5. Literatura 35

1. Uvod

U današnjem globalnom okruženju, gde se svet vrti oko digitalnih aplikacija koje su postale sastavni deo svakodnevnice, bezbednost podataka postaje suštinski važna. Baze podataka su, nažalost, izložene neprestanim napadima hakera, a ozbiljnost incidenata sa curenjem podataka i učestalost ovih pretnji neprestano rastu. Organizacije se suočavaju sa raznim klasama pretnji, uključujući phishing, ransomvare i krađu intelektualne svojine, koje godišnje beleže porast od više od 50%. U ovom kontekstu, bezbednost podataka, naročito u bazama podataka, postaje neizbežan prioritet za administratore.

U skladu s tim, MongoDB se ističe kao snažna NoSQL baza podataka prilagođena zahtevima savremenih aplikacija, sa istaknutim fokusom na bezbednost sistema. Ovaj rad istražuje različite aspekte bezbednosti u MongoDB, nudeći dubinski uvid u zahteve zaštite podataka i implementaciju bezbednosnih mehanizama.

Analizirajući bezbednosnu arhitekturu MongoDB skladišta podataka, ovaj rad prikazuje ključne komponente poput autentifikacije, autorizacije, revizije podataka i enkripcije. Autentifikacija se sprovodi kroz različite metode, uključujući SCRAM autentifikaciju, x.509 sertifikate, LDAP proxy autentifikaciju i Kerberos autentifikaciju. Praktični primeri autentifikacije dodatno ilustruju implementaciju ovih mehanizama u stvarnim situacijama.

Autorizacija putem ugrađenih uloga omogućava precizno upravljanje privilegijama korisnika, dok revizija podataka pruža mehanizme praćenja promena u sistemu. Enkripcija podataka, kroz transportnu, mirovnu i upotrebnu enkripciju, čini MongoDB sigurnim izborom za organizacije koje zahtevaju visok nivo zaštite podataka.

Praktični deo rada pruža konkretne smernice za implementaciju bezbednosnih mehanizama, demonstrirajući korake u autentifikaciji, autorizaciji, reviziji i enkripciji.

2. Zahtevi za obezbeđivanje podataka savremenih aplikacija

Tokom poslednjih deset godina narastao je broj pretnji, a zabrinutost za zaštitu privatnosti pojedinaca je postala izraženija. U odgovoru na ove izazove, različite industrije širom sveta započele su razne inicijative kako bi povećale nivo bezbednosti, smanjile prevaru i zaštitile lične informacije. Neka od tih inicijativa obuhvataju:

- Primenu PCI DSS standarda za upravljanje informacijama o korisnicima kartica.
- Primenjuju se HIPAA standardi za upravljanje zdravstvenim informacijama.
- GDPR i CCPA su usvojeni kako bi se osigurala zaštita privatnosti podataka o građanima EU i Kalifornije.
- FISMA se primenjuje radi osiguranja bezbednosti podataka u federalnoj vladi.
- FERPA se koristi kako bi se zaštitila privatnost evidencije o obrazovanju učenika.
- Sprovođenje prekogranične privatnosti u Aziji i Pacifiku postiže se putem aranžmana CPEA, koji stvara okvir za regionalnu saradnju u sprovođenju zakona o privatnosti. Pored ovih inicijativa, donose se i novi propisi svake godine da se nose sa novim pretnjama i novi zahtevi za strožim kontrolama koje regulišu korišćenje podataka. Svaki set propisa definiše bezbednost i reviziju zahteva koji su jedinstveni za određenu industriju ili aplikaciju, a usklađenost se procenjuje na osnovu projekta.

Važno je prepoznati da se usaglašenost može postići samo primenom kombinacije kontrola koje se mogu sažeti kao ljudi, procesi i platforme:

- Ljudi definišu specifične uloge, zaduženja i odgovornost
- Procesi definišu principe rada i poslovnu praksu
- Platforme definišu tehnologije koje se koriste za skladištenje i obradu podataka [1]

Slika 1. MongoDB arhitektura bezbednosti [1]

2.1. Bezbednosna arhitektura MongoDB skladišta podataka

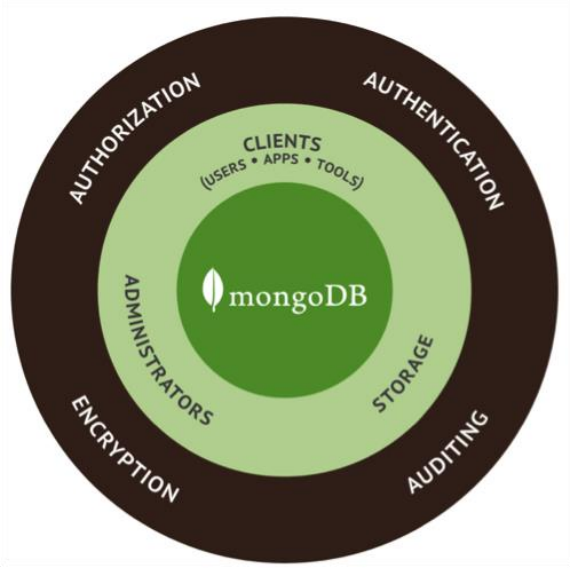
Iako postoje različite direktive koje regulišu različite aspekte, postoji nekoliko osnovnih zahteva koji su zajednički svim tim direktivama, uključujući:

- Ograničavanje pristupa podacima putem unapred definisanih privilegija i uloga.
- Implementiranje mera zaštite kako bi se sprečilo slučajno ili namerno otkrivanje, gubitak, uništavanje ili oštećenje ličnih podataka.
- Delegiranje dužnosti prilikom pristupa i obrade podataka.
- Vođenje evidencije aktivnosti korisnika, administrativnog osoblja i aplikacija sa bazom podataka.

Ovi zahtevi informišu bezbednosnu arhitekturu MongoDB-a, sa najboljim praksama za implementaciju bezbednog, usaglašenog okruženja za upravljanje podacima.

Holistička bezbednosna arhitektura mora da pokriva sledeće:

- Upravljanje pristupom korisnika radi ograničavanja pristupa osetljivim podacima, implementiran kroz kontrole **autentifikacije** i **autorizacije**
- **Revizija akcija baze podataka** za forenzičku analizu.
- **Zaštita podataka putem enkripcije** podataka u pokretu preko mreže, u upotrebi u bazi podataka i u stanju mirovanja u trajnom skladištu [1]



Slika 2. Holistička bezbednosna arhitektura MongoDB skladišta podataka [2]

2.2. Autentifikacija

Autentifikacija je ključan proces za potvrdu identiteta entiteta prilikom uspostavljanja veze sa MongoDB skladištem podataka. Kada je kontrola pristupa (autorizacija) omogućena, MongoDB zahteva od svih klijenata da se autentifikuju kako bi se utvrdio njihov pristup. MongoDB podržava različite mehanizme za autentifikaciju, uključujući:

- SCRAM
- Proveru autentičnosti sertifikata x.509
- LDAP proxy autentifikaciju
- Kerberos autentifikaciju

Ovi mehanizmi omogućavaju MongoDB da se integrira u postojeći sistem autentifikacije i zadovolji zahteve različitih okruženja. [1] , [3]

2.2.1. SCRAM autentifikacija

SCRAM (Salted Challenge Response Authentication Mechanism) predstavlja podrazumevani mehanizam autentifikacije za MongoDB. Kada korisnik pokuša da se autentifikuje, MongoDB primenjuje SCRAM kako bi proverio unete korisničke podatke u vezi sa korisničkim imenom, lozinkom i bazom podataka za autentifikaciju.

Primena SCRAM mehanizma za autentifikaciju u MongoDB donosi niz sigurnosnih karakteristika:

- Prilagodljiv faktor rada - Omogućava podešavanje faktora rada, što znači kontrolu nad brojem iteracija koje se koriste za hashiranje (šifrovanje) lozinke. Ova prilagodljivost pruža mogućnost optimizacije u skladu sa specifičnim sigurnosnim potrebama.
- Nasumične soli (engl. salt) po korisniku - Za svakog korisnika generiše se jedinstvena nasumična so (engl. salt) koja se koristi zajedno sa lozinkom. Ovo dodatno otežava potencijalnim napadačima da dešifruju lozinke korisnika, čineći autentifikaciju sigurnijom.
- Dvosmerna autentifikacija - SCRAM mehanizam omogućava dvosmernu autentifikaciju između servera i klijenta. Osim što klijent prolazi kroz autentifikaciju kod servera, server takođe prolazi kroz autentifikaciju kod klijenta. Ovo dodatno osigurava integritet i bezbednost komunikacije između njih.

- Svaka sesija SCRAM autentifikacije koristi nasumično generisane jednokratne brojeve kako bi se sprečilo ponovno izvršavanje napada. Ovo znači da čak i ako napadač presretne komunikaciju između klijenta i servera, neće moći ponovo koristiti te podatke za neovlašćeni pristup.

SCRAM mehanizam pruža pouzdanu zaštitu baze podataka od niza potencijalnih napada, uključujući:

- Prisluškivanje - SCRAM klijent nikada ne šalje lozinku kao običan tekst preko mreže, umesto toga koristi siguran protokol razmene podataka, čime sprečava napadača da uhvati i pročita komunikaciju između klijenta i servera.
- Ponovno izvršavanje (Replay) - SCRAM autentifikacija koristi nasumično generisane jednokratne brojeve za svaku sesiju, čime se invazivno ponovno izvršavanje onemogućava, jer su poruke protokola važeće samo tokom jedne sesije.
- Kompromitovanje baze podataka - SCRAM dodatno obezbeđuje "začinjavanje" i iterativno heširanje lozinki pre njihovog čuvanja u bazi podataka, otežavajući pristup i dešifrovanje čak i ako napadač pokuša pristupiti memoriji servera.
- Napad zlonamernog servera - SCRAM autentifikacija sprečava lažno predstavljanje napadača kao server prema klijentu, jer zahteva tačne SCRAM autentifikacione podatke klijenta za validaciju, čime dodatno pojačava bezbednost komunikacije.

Sve u svemu, SCRAM autentifikacija obezbeđuje pouzdanu zaštitu i odbranu od ovih vrsta potencijalnih napada, čime se osigurava sigurnost podataka i integritet MongoDB baze podataka.

Postupak razmene informacija kod SCRAM autentifikacije

Proces autentifikacije počinje kada klijent pošalje zahtev serveru za autentifikacijom. U ovom zahtevu, klijent obično šalje korisničko ime i random broj (Client Nonce). Nakon što server primi ovaj zahtev, odgovara klijentu porukom koja sadrži salt vrednost za određenog korisnika, broj iteracija, i Combine Nonce, koji se sastoji od konkatenacije Client Nonce i random broja koji generiše sam server (Server Nonce).

Kada klijent primi odgovor od servera, koristi dobijene informacije za izračunavanje Client Proof-a, koristeći iste korake koje je server koristio prilikom izračunavanja Stored Key. Osim toga, klijent računa i Client Signature na osnovu Combine Nonce i izračunatog Stored Key. Kada je ovaj proces završen, klijent šalje izračunate vrednosti (Client Proof i Client Signature) nazad serveru.

Nakon što server primi ove podatke od klijenta, koristi ih za proveru autentičnosti klijenta. Server želi da utvrdi da li se klijent zaista predstavlja kao onaj za koga tvrdi da jeste. Ako server utvrdi da su podaci koje je dobio validni, šalje svoj Server Signature nazad klijentu. Ovaj Server Signature je izračunat na osnovu Server Key-a i prethodno razmenjenih random brojeva.

Konačno, klijent koristi primljeni Server Signature kako bi proverio autentičnost servera i utvrdio da li je server zaista validan. Nakon ovog procesa, klijent i server su uspešno autentifikovani, omogućujući nastavak komunikacije između njih u sigurnom okruženju. [1], [4]

2.2.2. x.509 autentifikacija sertifikata

X.509 autentifikacija u MongoDB pruža pouzdan mehanizam za obezbeđenje komunikacije između klijenata i servera. Ova tehnologija omogućava sigurnu identifikaciju korisnika kroz upotrebu sertifikata, eliminišući potrebu za tradicionalnim korisničkim podacima, kao što su korisničko ime i lozinka.

Da bi se implementirala X.509 autentifikacija, neophodno je prvo generisati Autoritet za Sertifikate (CA). CA ima ključnu ulogu potpisivanja sertifikata, što garantuje njihovu validnost tokom celokupnog procesa autentifikacije. Nakon uspostavljanja CA, sledeći koraci obuhvataju generisanje sertifikata za MongoDB server i konfiguraciju servera da prihvati X.509 autentifikaciju. To uključuje podešavanje servera da koristi generisane sertifikate i prepozna CA kao validnog izdavača sertifikata.

Klijenti koji žele pristupiti MongoDB bazi podataka koriste X.509 autentifikaciju generišući svoje sertifikate, takođe potpisane od strane istog CA. Konfiguracija aplikacija ili alata koje klijenti koriste obuhvata podešavanje da koriste ove sertifikate prilikom autentifikacije. Nakon implementacije, sistem se testira kako bi se proverilo ispravno funkcionisanje X.509 autentifikacije.

Održavanje sigurnosti sistema uključuje redovno ažuriranje sertifikata, naročito dodavanje novih sertifikata za nove klijente koji žele pristup sistemu. Ovaj kontinuirani proces obezbeđuje visok nivo sigurnosti u MongoDB okruženju, pružajući pouzdanu autentifikaciju i bezbednu komunikaciju između čvorova baze podataka. Autentifikacija se vrši bez tradicionalnih korisničkih podataka, unapređujući sigurnost sistema i pružajući fleksibilnost u vezi sa vrstom i upotrebom sertifikata.

2.2.3. LDAP proxy autentifikacija

LDAP (Lightweight Directory Access Protocol) je protokol za pristup i održavanje direktorijuma, često korišćen za upravljanje identitetima, autentifikaciju i autorizaciju. U kontekstu MongoDB, LDAP se često primenjuje za centralizovano upravljanje korisničkim identitetima i pristupom.

Koristeći LDAP autentifikaciju u MongoDB-u, organizacije mogu centralizovano upravljati korisničkim pravima i pristupom resursima. Ovaj pristup donosi efikasnost i jednostavnost, posebno u okruženjima sa velikim brojem korisnika i kompleksnim strukturama prava pristupa.

Uvođenje LDAP autentifikacije u MongoDB zahteva nekoliko ključnih koraka. Prvi korak obuhvata konfiguraciju MongoDB servera da bi podržavao LDAP autentifikaciju. Ovde se postavljaju parametri kao što su LDAP URL, korisničko ime i lozinka za povezivanje sa LDAP serverom. Nakon uspešne konfiguracije, MongoDB server može izvršiti autentifikaciju korisnika putem LDAP-a. Tokom ovog procesa, korisnički podaci se proveravaju prema informacijama dostupnim na LDAP serveru. Ukoliko su podaci validni, korisniku se dozvoljava pristup MongoDB bazi podataka.

Važno je napomenuti da se postupak autentifikacije putem LDAP-a može prilagoditi specifičnim potrebama organizacije. Na primer, mogu se implementirati dodatne sigurnosne funkcionalnosti poput dvofaktorske autentifikacije ili postavljanje određenih privilegija zavisno od LDAP grupa kojima korisnik pripada.

LDAP autentifikacija u MongoDB donosi značajnu fleksibilnost i efikasnost u upravljanju identitetima i pravima pristupa u sistemu, čineći je popularnim izborom u mnogim organizacijama koje zahtevaju centralizovano i skalabilno rešenje za autentifikaciju. [1], [6]

2.2.4. Kerberos autentifikacija

Kerberos autentifikacija predstavlja izuzetno efikasan mehanizam za bezbednost pristupa u MongoDB okruženju. Ova tehnologija se često koristi kako bi se obezbedila pouzdana autentifikacija između klijenata i servera.

Kerberos autentifikacija je posebno korisna u scenarijima gde je bezbednost od suštinskog značaja. Kada se MongoDB konfiguriše za upotrebu Kerberosa, korisnici se autentifikuju putem sigurnih i kriptovanih *tiketa*, a čitav proces se odvija bez potrebe za slanjem

lozinki preko mreže. Ovo eliminiše potrebu za skladištenjem osetljivih lozinki u konfiguraciji, čime se povećava nivo bezbednosti.

Jedna od ključnih karakteristika Kerberos autentifikacije je sposobnost centralizovanog upravljanja identitetima. Kroz integraciju sa Kerberos distribuiranim sistemom autentifikacije, MongoDB omogućava organizacijama da održavaju jedinstvenu tačku upravljanja korisničkim identitetima, čime se pojednostavljuje proces upravljanja pravima pristupa i povećava kontrola nad bezbednošću sistema.

Postupak autentifikacije putem Kerberosa u MongoDB-u uključuje nekoliko ključnih koraka. Prvo se MongoDB server konfiguriše za prihvatanje Kerberos autentifikacije, određujući *principale*, tj. korisničke identitete. Klijenti, zatim, generišu Kerberos *tikete* kako bi se autentifikovali prema MongoDB serveru. Server potom validira tiket i dozvoljava pristup korisnicima u skladu sa definisanim pravilima.

Kerberos autentifikacija u MongoDB predstavlja snažan alat za jačanje bezbednosti i efikasnosti u sistemu. Njen doprinos u očuvanju integriteta autentifikacije, upravljanju identitetima i integraciji sa drugim sigurnosnim sistemima čini je privlačnim izborom za organizacije koje teže visokim standardima bezbednosti u radu sa MongoDB bazama podataka. [1], [7]

MongoDB nudi različite mehanizme autentifikacije prilagođene specifičnim potrebama organizacija. **SCRAM** autentifikacija se ističe kao jednostavna i efikasna opcija, pogodna za opšte aplikacije koje zahtevaju osnovno rešenje za autentifikaciju. Koristi korisnička imena i lozinke, a dodatna sigurnost je postignuta upotrebom soli (salt) prilikom čuvanja lozinki.

X.509 autentifikacija pruža viši nivo bezbednosti eliminisanjem potrebe za tradicionalnim korisničkim podacima. Ova tehnika koristi digitalne sertifikate za sigurnu autentifikaciju klijenata i servera i često se primenjuje u organizacijama koje zahtevaju dodatni sloj bezbednosti i kontrole pristupa.

LDAP Proxy autentifikacija pojednostavljuje integraciju MongoDB sa LDAP sistemima upravljanja identitetima. Kroz centralizovano upravljanje identitetima, omogućava efikasnu kontrolu pristupa bazama podataka, posebno korisna u organizacijama koje već koriste LDAP infrastrukturu za autentifikaciju.

Kerberos autentifikacija se izdvaja kao vrhunski izbor u bezbednosno osetljivim okruženjima. Ovaj mehanizam omogućava visok nivo bezbednosti kroz distribuirani sistem autentifikacije, preporučujući se organizacijama koje postavljaju visoke standarde u oblasti bezbednosti i traže centralizovano upravljanje identitetima u svom ekosistemu

2.3. Autorizacija

Sigurnosna autorizacija u MongoDB-u predstavlja proces utvrđivanja specifičnih privilegija koje određeni korisnik ima u bazi podataka. MongoDB primenjuje kontrolu pristupa baziranu na ulogama, poznatu kao RBAC (Role-Based Access Control), kako bi efikasno upravljao pristupom sistemu. Korisnici se pridružuju ulogama koje precizno definišu dozvole za pristup resursima i operacijama unutar baze podataka. Bez definisanih uloga, korisnicima je zabranjen pristup sistemu.

Uloge su ključne u dodeljivanju ovlašćenja, gde privilegije mogu biti jasno definisane unutar uloga ili nasleđene od drugih uloga. MongoDB pruža unapred definisane uloge poput dbAdmin, dbOwner, clusterAdmin, readWrite, i mnoge druge, pojednostavljujući upravljanje pristupom i privilegijama. Ove uloge su detaljno opisane u poglavlju 2.3.1.

Ono što čini MongoDB moćnim je mogućnost prilagođavanja uloga putem korisnički definisanih uloga. Ovo omogućava precizno dodeljivanje privilegija korisnicima prema njihovim specifičnim potrebama za pristupom podacima.

Upravljanje i održavanje korisničkih naloga je pojednostavljeno delegiranjem uloga između timova, omogućavajući doslednu primenu sigurnosnih politika. MongoDB takođe omogućava detaljno definisanje korisničkih privilegija na nivou baze podataka i kolekcija, pružajući visok stepen granularnosti u upravljanju pristupom podacima.

Privilegije se definišu na nivou resursa i akcija, gde resurs može biti baza podataka, zbirka, skup kolekcija ili klaster. Akcije određuju dozvoljene operacije nad resursom. Kada se korisnicima dodeljuju uloge, automatski dobijaju sve privilegije vezane za tu ulogu, omogućavajući precizno upravljanje pristupom i privilegijama.

Prilikom dodele uloga, prvi korisnik u bazi podataka obično treba biti administrator korisnika sa privilegijama za upravljanje drugim korisnicima.

MongoDB pruža i ugrađene uloge koje pokrivaju česte potrebe, ali takođe omogućava kreiranje i modifikaciju korisnički definisanih uloga. Različite klase korisnika i aplikacija, kao i različite uloge administratora, mogu imati različite privilegije, što omogućava jasno razdvajanje dužnosti unutar organizacije.

Sve operacije i promene na nivou uloga i privilegija mogu se pratiti putem mogućnosti revizije dostupnih u MongoDB Enterprise Advanced. Ova mogućnost omogućava organizacijama da održavaju potpunu operativnu kontrolu i obezbede transparentnost u svim akcijama u cilju usklađenosti i izveštavanja. [1], [8]

2.3.1. Ugrađene uloge

Kao što je gore napomenuto u sistemu MongoDB, pristup podacima i komandama se ostvaruje putem sistema autorizacije zasnovanog na ulogama. MongoDB već uključuje ugrađene uloge koje nude različite nivoe pristupa koji su često potrebni u kontekstu baza podataka. Osim toga, korisnicima se omogućava kreiranje sopstvenih uloga prilagođenih specifičnim zahtevima sistema.

Svaka uloga u MongoDB-u definiše skup privilegija koji omogućava izvođenje određenih radnji nad definisanim resursima. Važno je napomenuti da svaka uloga ima uticaj na bazu podataka u kojoj je definisana i može odobriti pristup na nivou kolekcija sa visokim stepenom granularnosti.

U kontekstu ugrađenih uloga MongoDB-a, svaka od njih precizno definiše nivoe pristupa na nivou baze podataka za sve nesistemske kolekcije unutar te baze podataka, dok za sistemske kolekcije definiše pristup na nivou kolekcija. [9]

Svaka baza podataka u MongoDB sistemu ima:

- Uloge korisnika (Database User Roles)
- Uloge za administraciju baze podataka (Database Administration Roles)
- Administratorske uloge za upravljanje klasterom (Cluster Administration Roles)
- Administratorske uloge za upravljanje procesima oporavka baze podataka (Backup/Restore Roles)

Ove uloge igraju ključnu ulogu u pristupu i upravljanju bazom podataka, obezbeđujući kontrolu nad korisnicima i njihovim privilegijama.

Uloge korisnika (Database User Roles)

MongoDB uključuje uloge korisnika baze podataka koje pružaju različite nivoe pristupa unutar svake baze. Na primer, uloga "**read**" omogućava čitanje podataka u svim nesistemskim kolekcijama, kao i u kolekciji "system.js". Ova uloga omogućava niz radnji, uključujući pristup podacima i statistikama kolekcija.

Uz ulogu "read", postoji i uloga "**readWrite**" koja ide korak dalje i omogućava izmenu podataka u nesistemskim kolekcijama, uključujući kolekciju "sistem.js". Ova uloga pruža širok spektar privilegija, uključujući dodavanje, izmenu i brisanje podataka, upravljanje kolekcijama i indeksima, kao i druge ključne operacije nad podacima.

Važno je napomenuti da MongoDB omogućava kreiranje sopstvenih uloga koje se prilagođavaju specifičnim potrebama sistema. Ove uloge mogu se dodatno podešavati kako bi odgovarale konkretnim zahtevima za pristupom i upravljanjem podacima. Ova fleksibilnost omogućava precizno upravljanje privilegijama i sigurnosnim politikama unutar sistema. [9]

Uloge za administraciju baze podataka (Database Administration Roles)

Ulogama administracije baze podataka dodeljuju se ključne uloge u upravljanju pojedinim bazama podataka unutar sistema MongoDB. Svaka baza podataka uključuje sledeće uloge administracije baze podataka:

- **dbAdmin** - Ova uloga omogućava obavljanje administrativnih zadataka vezanih za konfiguraciju baze podataka, šemu, indeksiranje i prikupljanje statistike. Važno je napomenuti da ova uloga ne daje privilegije za upravljanje korisnicima i ulogama.
- **dbOwner** - Vlasnik baze podataka ima širok spektar privilegija i može izvršiti bilo koju administrativnu radnju na bazi podataka. Ova uloga objedinjuje privilegije koje obuhvataju uloge "readWrite," "dbAdmin," i "userAdmin." Time omogućava potpunu kontrolu nad bazom podataka.
- **userAdmin** - Ova uloga omogućava kreiranje i izmenu uloga i korisnika unutar trenutne baze podataka. Osim toga, userAdmin uloga omogućava korisnicima da dodele privilegije drugim korisnicima, uključujući sebe. To znači da ova uloga, indirektno, pruža superkorisnički pristup bazi podataka. Ako se koristi na nivou admin baze podataka, userAdmin uloga omogućava i upravljanje čitavim klasterom. Osim navedenih privilegija, userAdmin uloga eksplicitno obezbeđuje i druge radnje kao što su promena lozinke, kreiranje i brisanje uloga i korisnika, dodeljivanje i opozivanje uloga, kao i postavljanje restrikcija za autentikaciju. Ovo pruža potpunu kontrolu nad upravljanjem korisnicima i ulogama unutar baze podataka. [9]

Administratorske uloge za upravljanje klasterom (Cluster Administration Roles)

U kontekstu administratorskih uloga za upravljanje klasterom, MongoDB nudi niz ugrađenih uloga koje su ključne za efikasno rukovođenje klasterom. Ove uloge obuhvataju:

- **clusterManager** - Ova uloga pruža privilegije kojima se omogućava opšte upravljanje i nadzor nad klasterom. To uključuje kontrolu nad konfiguracijom i performansama klastera.

- **clusterMonitor** - Uloga clusterMonitor je fokusirana isključivo na privilegije za čitanje podataka u svrhu nadzora klastera. Ova uloga omogućava korisnicima da prate stanje i performanse klastera bez mogućnosti izmene konfiguracije.
- **hostManager** - HostManager uloga omogućava upravljanje serverima koji čine klaster. To uključuje kontrolu nad dodavanjem, uklanjanjem i konfiguracijom čvorova u klasteru.
- **clusterAdmin** - Ova uloga objedinjuje sve privilegije prethodno navedenih uloga i pruža potpunu kontrolu nad upravljanjem klasterom. Korisnicima sa ovom ulogom daju se široka ovlašćenja za rukovođenje i nadzor svih aspekata klastera. [9]

Administratorske uloge za upravljanje procesima oporavka baze podataka (Backup/Restore Roles)

Pored administratorskih uloga za upravljanje klasterom, MongoDB takođe nudi uloge dizajnirane za upravljanje procesima oporavka baze podataka. Te uloge uključuju:

- **backup** - Ova uloga omogućava minimalne privilegije za kreiranje rezervnih kopija podataka. Korisnicima sa ovom ulogom omogućava se izrada sigurnosnih kopija podataka radi zaštite od gubitka.
- **restore** - Uloga "restore" pruža privilegije za oporavak baze podataka iz prethodno kreiranih rezervnih kopija. Ovo je ključno za obnovu podataka u slučaju problema ili gubitka podataka.

Ove uloge za oporavak baze podataka igraju važnu ulogu u očuvanju sigurnosti podataka i omogućavaju efikasno upravljanje procesima oporavka. Sve ove uloge zajedno čine MongoDB sistem veoma prilagodljivim i sigurnim za rukovanje različitim aspektima upravljanja i oporavka baze podataka. [9]

Korisnicima koji su kreirani u admin bazi dodeljene su specifične uloge koje im pružaju privilegije nad svim bazama, osim "config" i "local":

- **readAnyDatabase** - Pruža privilegije čitanja nad svim bazama
- **readWriteAnyDatabase** - Slična je ulozi readWrite, ali važi za sve baze
- **userAdminAnyDatabase** - Pruža iste privilegije kao userAdmin, ali važi za sve baze
- **dbAdminAnyDatabase** - Pruža iste privilegije kao dbAdmin, ali važi za sve baze

2.4. Revizija podataka

Revizija kod MongoDB Enterprise uključuje mogućnost praćenja aktivnosti mongod i mongos instanci. Ova funkcionalnost omogućava administratorima i korisnicima praćenje aktivnosti sistema u okruženjima sa više korisnika i aplikacija.

Mehanizam za reviziju može zapisivati događaje u konzoli, syslog-u, JSON fajlu ili BSON fajlu. Da bi se omogućila a revizija, potrebno je postaviti destinaciju za audit izlaz pomoću – *auditDestination* u konfiguracionom fajlu.

Jednom omogućen, sistem za reviziju može zabeležiti sledeće operacije:

- Šema (DDL),
- Replica set i sharded cluster,
- Autentikacija i autorizacija, i
- CRUD operacije (uz postavljanje auditAuthorizationSuccess na true).

Počevši od MongoDB verzije 5.0, sekundarne replike ne beleže DDL audit događaje za replikovane promene. DDL audit događaji i dalje se beleže za DDL operacije koje menjaju lokalnu bazu podataka i kolekciju system.profile.

Pomoću sistema za reviziju, mogu se postaviti filteri kako bi se ograničile zabeleženi događaji.

Sistem za auditing zapisuje svaki audit događaj u memorijski bafer audit događaja. MongoDB periodično zapisuje ovaj bafer na disk. Za događaje prikupljene sa jedne konekcije, događaji imaju ukupan redosled, ako MongoDB zapiše jedan događaj na disk, sistem garantuje da su svi prethodni događaji za tu konekciju zapisani na disk.

Ako audit događaj odgovara operaciji koja utiče na trajno stanje baze podataka, kao što je modifikacija podataka, MongoDB će uvek prvo zapisati audit događaj na disk pre nego što zapiše u journal za tu operaciju. Drugim rečima, pre dodavanja operacije u journal, MongoDB zapisuje sve audit događaje na konekciji koja je pokrenula operaciju, uključujući i samu operaciju.

Revizija podataka u MongoDB doprinosi bezbednosti, transparentnosti i olakšava upravljanje i analizu istorije podataka. Ovaj mehanizam se često koristi u situacijama gde je važno pratiti promene, očuvati integritet podataka i obezbediti sredstva za oporavak u slučaju grešaka ili neželjenih promena. [1], [10]

2.5. Enkripcija podataka

Današnji svet funkcioniše na temelju podataka. Podaci koje organizacija kreira, čuva i menja su njeni privredni resursi. Šifrovanje podataka predstavlja ključni korak za sve organizacije, budući da omogućava zaštitu podataka, korisnika i usaglašenost sa propisima o privatnosti podataka. Organizacija mora da osigura da njeni interni korisnici imaju pristup samo relevantnim podacima. Organizacija treba da očuva svoje podatke od mogućih kompromisa i neovlašćenog pristupa.

Posebno za organizacije koje operišu sa osetljivim informacijama, potreba za šifrovanjem podataka postaje još važnija. U nastavku su objašnjeni razlozi zbog kojih je šifrovanje podataka važno, uprkos postojanju snažnih mera autentifikacije i autorizacije, i različiti aspekti šifrovanja u MongoDB bazi podataka, kao i najbolje prakse u vezi sa šifrovanjem. [11]

Zašto je važno šifrovanje podataka?

Izvrsno sredstvo za sprečavanje zloupotrebe privilegija baze podataka je kontrola pristupa zasnovana na ulogama (RBAC). Međutim, važno je napomenuti da ova funkcionalnost samo zamagljuje određene ranjivosti koje se moraju uzeti u obzir prilikom osiguravanja sistema. RBAC ne adresira situacije u kojima zlonamerni akteri njuškaju mrežu, direktno ciljaju podatke ili pokušavaju pristupiti memoriji hosta baze podataka.

Jednostavno rečeno, šifrovanje je čin kodiranja podataka kako bi se učinio nečitljivim za nekoga ko nema ovlašćenje da pristupi podacima. Samo ovlašćena lica koja imaju *ključ* za šifrovanje mogu da čitaju i razumeju podatke. Efikasno primenjena metoda šifrovanja treba da obezbedi punu sigurnost od kompromitovanog pristupa koji vodi do ukradenih podataka.

Šifrovanje na osnovnom nivou radi tako što zamenjuje slova, brojeve i simbole drugim znacima kako bi stvorilo šifru. Šifra predstavlja kolekciju znakova koja reprezentuje originalnu poruku i, u ovom slučaju, izvorne podatke. Kreator ove šifre čuva ključ za šifrovanje, broj koji definiše način šifriranja poruke. Moderni postupci šifrovanja su znatno kompleksniji od jednostavne zamene karaktera. Većina savremenih implementacija preuzima običan tekst i transformiše ga u šifrovanu verziju sastavljenu od delova ili tokova bajtova, koji se može pojaviti u obliku teksta, slike ili binarnog blob-a. Bez pristupa ovom ključu, ni računar ni osoba ne mogu dešifrovati stvarne podatke, a neovlašćeni korisnici ne mogu ispravno da interpretiraju informacije. [11]

Tipovi šifrovanja

Metode šifrovanja se obično svrstavaju u dve osnovne kategorije: simetrično šifrovanje i asimetrično šifrovanje:

- Simetrično šifrovanje - metoda u kojoj se isti ključ koristi kako za šifrovanje tako i za dešifrovanje (ključ je simetričan).
- Asimetrično šifrovanje - Ova metoda, takođe poznata kao kriptografija javnog ključa, koristi dva različita ključa: privatni ključ i javni ključ, koji nisu isti. Javni ključ može biti deljen sa bilo kim, dok je privatni ključ dostupan samo ovlašćenim korisnicima. Javni ključ se koristi za šifrovanje podataka, a ti podaci mogu biti dešifrovani isključivo uz pomoć privatnog ključa.

Obe metode imaju svoje slabosti, koje variraju u zavisnosti od prirode podataka, ali istovremeno zasenjuju potencijalne propuste koje RBAC može da ostavi, kao što su njuškanje mreže i krađa podataka. Zbog ove potrebe za osiguranjem, MongoDB primenjuje obe ove metode u svojim bazama podataka, u zavisnosti od specifičnih karakteristika podataka. [11]

Administratori mogu da šifruju MongoDB podatke u prenosu preko mreže i u mirovanju u trajnom skladištu i rezervnim kopijama. Korisnici mogu da šifruju podatke na nivou terena štiteći osetljive informacije od administratora i drugih legitimnih korisnika dok se podaci koriste na serveru.

2.5.1. Enkripcija podataka u prenosu (Transport Encryption)

Šifrovanje podataka u prenosu predstavlja zaštitu podataka tokom kretanja sa jednog mesta na drugo. Pre nego što budu poslati, podaci se enkriptuju kako bi bili sigurni tokom svog "putovanja", a zatim se dešifruju i verifikuju na odredištu.

MongoDB Atlas pruža podršku za sigurnost na nivou transportnog sloja (TLS - transport layer security) i sigurnosnog sloja soketa (SSL - secure socket layer) kako bi obezbedio komunikaciju između aplikacije i servera, kao i komunikaciju unutar klastera, postavljanjem sertifikata za servere. Sertifikati predstavljaju elektronske dokumente koji potvrđuju validnost vlasništva nad privatnim ključem. Ovi sertifikati osiguravaju da se podaci enkriptuju tokom transporta preko sigurne mrežne veze, sprečavajući bilo kakve pokušaje njuškanja mreže, kao što su njuškanje paketa ili lažiranje IP/DNS adresa.

Kada se uzme u obzir upotreba javnog ključa, može se primetiti da MongoDB primenjuje asimetrično šifrovanje kako bi osigurao podatke tokom njihovog prenosa. U suprotnom, kod

simetričnog šifrovanja, morala bi postojati komunikacija i razmena ključeva između strana putem nekog komunikacionog kanala, što bi podrazumevalo da ključ ostane siguran tokom tog prenosa i stvara potencijalnu tačku ranjivosti.

Korišćenjem javnog ključa za šifrovanje, samo namenski primalac može dešifrovati poruku pošiljaoca koristeći svoj privatni ključ. Prednost kriptografije javnog ključa u prenosu je da ne postoji potreba za deljenjem privatnih ključeva, dok korišćenje potpisa omogućava primalcu da proveri autentičnost poruke i njenog pošiljaoca.

Mane asimetrične enkripcije uključuju sporiju brzinu od simetrične enkripcije i potrebu za većom potrošnjom resursa. Ako podaci nisu aktivno premešteni sa uređaja na uređaj ili sa mreže na mrežu, tada nije neophodno praviti ovakav kompromis i simetrično šifrovanje pruža dovoljan nivo sigurnosti.

MongoDB primenjuje dodatne sigurnosne mere kako bi se zaštitio od potencijalnih rizika asimetrične enkripcije. Jedan od tih mehanizama je korišćenje *Forward Secrecy cypher* paketa. Ovi paketi generišu efemerni ključ sesije koji je zaštićen privatnim ključem servera. Bitno je to što ovaj ključ sesije nikada nije prenesen između strana, što osigurava da čak i u slučaju kompromitacije privatnog ključa servera, prethodne sesije ne mogu biti dešifrovane pomoću kompromitovanog ključa. MongoDB podržava sledeće algoritme:

- Ephemeral Diffie-Hellman (DHE)
- Ephemeral Elliptic Curve Diffie-Hellman (ECDHE)

Ovi algoritmi osiguravaju da efemerni ključevi sesije budu generisani za svaku novu sesiju, dodatno povećavajući sigurnost i sprečavajući pristup prošlim sesijama čak i u slučaju ugroženog privatnog ključa servera. [11], [12]

2.5.2. Enkripcija podataka u mirovanju (Encryption at Rest)

Šifrovanje podataka u stanju mirovanja osigurava podatke koji su neaktivni, odnosno nisu u prenosu. Ovaj statički režim eliminiše rizik prenosa podataka koji se rešava korišćenjem asimetrične enkripcije, čineći simetričnu enkripciju pouzdanom metodom.

Podaci na disku ostaju šifrovani, osim ako napadač ima pristup ključu za dešifrovanje. Šifrovanje u stanju mirovanja pruža zaštitu osetljivih podataka u slučaju gubitka ili krađe fizičkih diskova ili otkrivanja datoteka u bazi podataka. Ovo stvara dodatni sloj sigurnosti za bazu podataka, što znači da pristup pisanim datotekama za skladištenje mogu imati samo ovlašćeni procesi ili aplikacije kada ih dešifruju.

MongoDB Atlas automatski uključuje šifrovanje u stanju mirovanja za diskove na svakom čvoru u klasteru kao podrazumevano podešavanje. Takođe ima opciju dodatnog aktiviranja šifrovanja putem WiredTiger mehanizma za skladištenje. Ova mogućnost omogućava da se koristi usluga upravljanja ključevima (KMS) odabranog dobavljača cloud usluga. KMS predstavlja servis koji centralizuje upravljanje svim ključevima za šifrovanje.

MongoDB podržava KMS usluge od strane AWS, Azure i Google Cloud Platform. Ova karakteristika je izuzetno korisna za one koji žele da preuzmu kontrolu nad vlasništvom nad generisanim ključevima za šifrovanje i ne žele se oslanjati samo na ključeve za šifrovanje koje nudi MongoDB. [11]

Proces šifrovanja podataka u MongoDB-u obuhvata niz ključnih koraka kako bi se osigurala potpuna zaštita podataka. Ovaj postupak uključuje:

- Generisanje glavnog (master) ključa - Prvi korak je kreiranje glavnog ključa, koji će poslužiti kao osnov za šifrovanje podataka.
- Generisanje ključeva za svaku bazu podataka - Za svaku bazu podataka u MongoDB-u generišu se jedinstveni ključevi, što omogućava izolaciju i sigurnost podataka na nivou baza podataka.
- Šifrovanje podataka pomoću ključeva baze podataka - Nakon što su generisani ključevi za svaku bazu podataka, podaci u tim bazama se šifruju pomoću odgovarajućih ključeva. Ovaj korak garantuje da podaci ostanu sigurni i nečitljivi bez odgovarajućeg ključa za dešifrovanje.
- Šifrovanje ključeva baze podataka pomoću glavnog ključa - Kako bi se dodatno pojačala sigurnost, ključevi baze podataka se takođe šifruju pomoću glavnog ključa. Ovo dodatno otežava neovlašćeni pristup podacima.

Važno je napomenuti da se šifrovanje odvija transparentno u sloju za skladištenje. To znači da su sve datoteke sa podacima na disku potpuno šifrovane iz perspektive sistema datoteka. Sama MongoDB baza podataka upravlja dešifrovanjem podataka samo kada su potrebni za obradu, dok podaci postoje samo u nešifrovanom stanju u memoriji tokom njihove upotrebe. Ovo osigurava da podaci budu sigurni u svim fazama rukovanja podacima, uključujući i tokom prenosa između servera i klijenta. [13]

Ključevi baze podataka su strogo interni za MongoDB server i čuvaju se na disku isključivo u šifrovanom obliku. Važno je napomenuti da glavni ključ nikada ne bude zapisan na disk, bez obzira na okolnosti. Sam glavni ključ je jedini spoljni element u sistemu (odnosno, čuva se odvojeno od ključeva baze podataka i samih podataka) i zahteva spoljno upravljanje.

Što se tiče upravljanja glavnim ključem, MongoDB-ov šifrovani mehanizam za skladištenje pruža dve opcije:

- Integracija sa uređajem za upravljanje ključevima treće strane putem protokola za interoperabilnost upravljanja ključevima (KMIP). Ovo se preporučuje kao optimalna opcija.
- Lokalno upravljanje ključevima putem datoteke ključeva.

Na ovaj način, MongoDB omogućava efikasno i sigurno upravljanje ključevima, čime se obezbeđuje integritet i pouzdanost sistema šifrovanja podataka. [13]

2.5.3. Enkripcija podataka u upotrebi (In-use Encryption)

U MongoDB-u, podaci prolaze kroz snažan proces šifrovanja na strani klijenta, koristeći ključeve za šifrovanje koje kontroliše sam korisnik. Ovaj postupak osigurava da se ostljivi podaci šifruju pre slanja u bazu podataka, prilikom skladištenja unutar nje i prilikom preuzimanja. Prednosti ovog pristupa su značajne i uključuju:

- Podaci šifrovani tokom celog životnog ciklusa - Ovaj pristup predstavlja najjaču tehničku kontrolu, čime se osigurava da podaci ostaju šifrovani tokom čitavog svog životnog ciklusa. To znači da su podaci bezbedni kako tokom upotrebe, tako i u rezervnim kopijama, u stanju mirovanja i tokom prenosa između različitih delova sistema.
- Brži ciklus razvoja aplikacija - MongoDB olakšava razvoj aplikacija za obradu osetljivih podataka. Programeri više ne moraju biti eksperti za bezbednost ili kriptografiju kako bi ugradili šifrovanje u svoje aplikacije. To znači brži razvoj i implementaciju bezbednih aplikacija.
- Rešavanje kritičnih slučajeva privatnosti podataka - Ovaj pristup omogućava klijentima da ispune stroge zahteve za zaštitu privatnosti podataka, kao što su HIPAA, GDPR, PCI, CCPA i mnogi drugi standardi i zakoni o zaštiti podataka.

MongoDB nudi dve ključne funkcionalnosti za šifrovanje kako bi se zadovoljile različite potrebe zaštite podataka:

- Client-Side Field Level Encryption (CSFLE) - Ova funkcija omogućava šifrovanje na nivou pojedinih polja unutar dokumenata, pružajući granularnu kontrolu nad šifrovanim sadržajem. Na ovaj način mogu se šifrovati samo određeni delovi podataka koji zahtevaju dodatnu zaštitu.

- Queryable Encryption (Queryable enkripcija) - Ova funkcija omogućava postavljanje upita nad šifrovanim podacima i dohvatanje šifrovanih rezultata, bez potrebe za dešifrovanjem podataka pre izvođenja upita. To čini analizu i pretragu šifrovanih podataka jednostavnom i efikasnom. [14]

Enkripcija na nivou polja na strani klijenta (CSFLE)

MongoDB-ova tehnologija Client-Side Field Level Encryption (CSFLE) predstavlja mehanizam za enkripciju podataka na nivou polja, direktno implementiran na strani klijenta. Ovakva enkripcija omogućava preciznu kontrolu korisnicima nad tim koje polja u kolekciji žele da budu enkriptovana i na koji način. Ključna prednost ovog pristupa leži u tome što za svako polje može biti specificiran jedinstveni ključ za enkripciju, što pruža izuzetnu fleksibilnost u zaštiti podataka pre nego što oni dospeju na server.

Bitan pojam kod CSFLE je **šema enkripcije** koja predstavlja JSON objekat koji detaljno definiše koja polja će biti enkriptovana i na koji način. Unutar svakog enkripcionog pravila u šemi, definišu se informacije o algoritmu za enkripciju, ključu za enkripciju i BSON tipu polja koje će biti enkriptovano. Postoje dva različita algoritma za enkripciju deterministički i randomiziran. Deterministička enkripcija omogućava izvođenje upita nad enkriptovanim poljima jer uvek generiše isti izlaz za isti ulaz. S druge strane, randomizirani algoritam, koji se koristi kada nije planirano izvršenje upita na enkriptovanom polju, pruža različite izlaze za iste ulaze.

U samom postupku enkripcije, ključevi igraju ključnu ulogu. **Customer Master Key (CMK)** koristi se za enkripciju ključeva **Data Encryption Key (DEK).** CMK predstavlja najosetljiviji podatak u celom CSFLE sistemu, budući da njegovo otkrivanje može ugroziti sigurnost svih podataka. DEK ključevi se, zatim, koriste za enkripciju polja u MongoDB dokumentima, a čuvaju se enkriptovani u **Key Vault** kolekciji mongo baze.

Naglašava se da Key Vault kolekcija mora biti indeksirana po imenima ključeva, koja sadrže informaciju o polju koje se enkriptuje. Preporučuje se korišćenje provajdera za upravljanje ključevima (KMS) kao dodatni sloj sigurnosti, poput Amazon Web Services KMS-a, Azure Key Vault-a ili Google Cloud Platform KMS-a. Ova kombinacija elemenata pruža sveobuhvatan pristup enkripciji polja u MongoDB, obezbeđujući visok nivo sigurnosti podataka tokom celog procesa enkripcije, uključujući čuvanje ključeva u odvojenom skladištu.

Queryable enkripcija

Queryable enkripcija (Queryable Encryption) u MongoDB-u pruža niz naprednih funkcionalnosti koje omogućavaju efikasno upravljanje i upite nad šifrovanim podacima, a istovremeno održava najviši nivo sigurnosti i privatnosti. Ove funkcionalnosti obuhvataju:

- Šifrovanje polja osetljivih podataka sa strane klijenta - Osetljivi podaci se šifruju na strani klijenta pre nego što budu poslati na server baze podataka. Ovo osigurava da su podaci zaštićeni pre samog skladištenja.
- Čuvanje polja osetljivih podataka kao potpuno nasumičnih šifrovanih podataka na strani servera baze podataka - Na serveru baze podataka, polja osetljivih podataka čuvaju se kao potpuno nasumični šifrovani podaci. Ovaj pristup čini šifrovanje transparentnim za bazu podataka, koja ne mora biti svesna specifičnih podataka.
- Pokretanje ekspresivnih upita za šifrovane podatke - MongoDB omogućava izvođenje upita nad šifrovanim podacima bez potrebe za dešifrovanjem tih podataka. To uključuje pretrage jednakosti, kao i naprednije upite poput opsega, prefiksa, sufiksa i podnizova, koji su planirani za buduća izdanja.

Svi ovi zadaci se obavljaju bez znanja servera baze podataka o specifičnim podacima koje obrađuje. Osetljivi podaci ostaju šifrovani tokom celog svog životnog ciklusa, uključujući transport, mirovanje, upotrebu, evidenciju i rezervne kopije. Dešifrovanje se vrši isključivo na strani klijenta, čime se osigurava da samo ovlašćene osobe imaju pristup ključevima za dešifrovanje.

Za konfiguraciju enkripcije za upite, MongoDB pruža sledeće mehanizme:

- Automatsko šifrovanje - Ova opcija omogućava izvođenje šifrovanih operacija čitanja i pisanja bez potrebe za pisanjem dodatnog koda za definisanje načina šifrovanja polja. MongoDB automatski upravlja šifrovanjem, što je korisno za pojednostavljenje razvoja aplikacija i obezbeđivanje zaštite podataka bez dodatne složenosti.
- Eksplicitno šifrovanje - Ovaj pristup pruža veću kontrolu nad šifrovanjem i omogućava programerima da izvode šifrovane operacije čitanja i pisanja putem biblioteke za šifrovanje dostupne u MongoDB drajverima. Programeri moraju dodatno definisati logiku za šifrovanje kroz ovu biblioteku u celoj aplikaciji. Ovo je korisno kada postoje specifični zahtevi za šifrovanje ili složeniji scenariji upravljanja ključevima.

Sve ove karakteristike čine Queryable enkripciju izuzetno svestranom i snažnom funkcionalnošću za zaštitu i upravljanje osetljivim podacima u MongoDB-u. [16]

3. Praktični deo

3.1. Autentifikacija

Za potvrdu autentičnosti klijenata u MongoDB-u, potrebno je kreirati odgovarajućeg korisnika. Korisnik se može dodati pomoću metode *db.createUser()* u mongo shell-u. Prvi korisnik koji se kreira mora imati privilegiju za stvaranje drugih korisnika, a uloge *userAdmin* ili *userAdminAnyDatabase* omogućavaju tu privilegiju.

Neophodno je kreirati administratora korisnika. Koristeći mongosh, prvo je potrebno preći na admin bazu podataka komandom *use admin*, a zatim dodati *myUserAdmin* korisnika sa ulogama *userAdminAnyDatabase,* koja omogućava kreiranje korisnika u svima bazama *i readWriteAnyDatabase*, koja omogućava čitanje i pisanje podataka u bazi, komandom *db.createUser()* kao na slici 3. Funkcija *createUser* kao parametre uzima *user* - korisničko ime korisnika, *pwd*- šifru korisnika i *roles* – uloge koje se dodeljuju korisniku. [17], [18], [19]

```
Enterprise test> use admin
switched to db admin
Enterprise admin> db.createUser(
...   {
...     user: "myUserAdmin",
...     pwd: "userAdmin123",
...     roles: [
...       { role: "userAdminAnyDatabase", db: "admin" },
...       { role: "readWriteAnyDatabase", db: "admin" }
...     ]
...   }
... );
{ ok: 1 }
Enterprise admin>
```

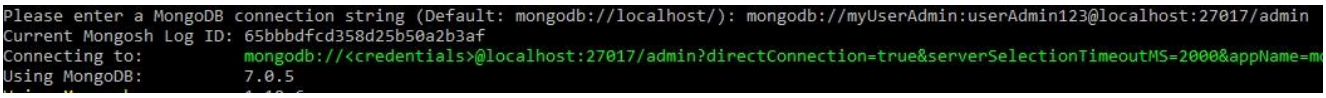
Slika 3. Kreiranje administratora korisnika

Svaki korisnik može se ulogovati u MongoDB bazu podataka kao autentifikovani korisnik na nekoliko načina. Prva dva načina se odvijaju tokom kreiranja konekcije sa bazom.

Prvi način je autentifikacija putem URI-ja. Na slici 4 prikazana je autentifikacija *myUserAdmin* korisnika putem URI-ja gde su detalji o korisničkom imenu i lozinci integrisani u URI.

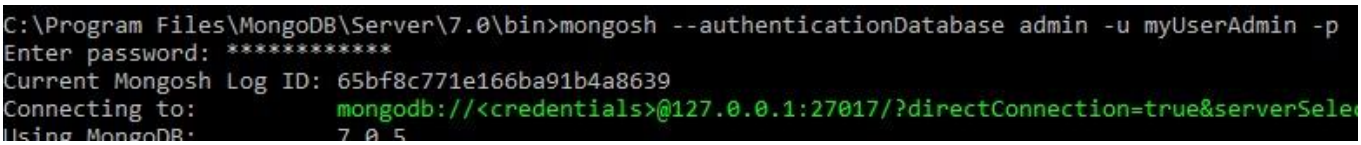
URI počinje s *mongodb://,* što označava da se koristi MongoDB protokol. Naredni deo, *myUserAdmin:userAdmin123*, predstavlja korisničko ime *myUserAdmin* i lozinku *userAdmin123*

razdvojene dvotačkom. Ovaj segment URI-ja pruža MongoDB serveru potrebne autentifikacione podatke. Nakon toga, simbol @ deli korisničke podatke od adrese servera. *localhost:27017* predstavlja adresu i port MongoDB servera. U ovom slučaju, MongoDB server je pokrenut lokalno na portu 27017. Poslednji deo URI-ja*, /admin*, predstavlja putanju do baze podataka na kojoj će se vršiti autentifikacija. U ovom slučaju, korisnik *myUserAdmin* se autentifikuje na *admin* bazi podataka.



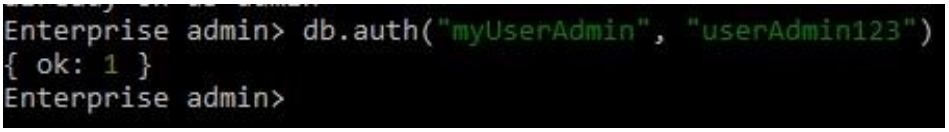
Slika 4. Autentifikacija myUserAdmin korisnika putem URI-ja

Drugi način je prikazan na slici 5, gde*, --authenticationDatabase admin* označava bazu podataka za autentifikaciju*, -u myUserAdmin* specificira korisničko ime, dok *-p* ukazuje da će biti potrebno uneti lozinku nakon izvršavanja komande.



Slika 5. Autentifikacija myUserAdmin korisnika

Treći način je korišćenjem *db.auth* komande u mongosh komandnoj liniji, kao na slici 6. Prvo je neophodno preći na admin bazu podataka, a zatim komandom *db.auth(user, pwd)* uneti username i password tog korisnika.



Slika 6. Autentifikacija korisnika korišćenjem db.auth()

3.2. Autorizacija

Kao što je u poglavlju 2.3. objašnjeno, MongoDB primenjuje kontrolu pristupa zasnovanu na ulogama, poznatu kao RBAC (Role-Based Access Control). Svaki korisnik se pridružuje jednoj ili više uloga koje precizno definišu njihovu dozvolu za pristup resursima i operacijama u okviru baze podataka. U ovom delu rada biće kreirano nekoliko tipova korisnika sa različitim privilegijama.

3.2.1. Kreiranje readOnly korisnika i demonstracija njegovih privilegija

U poglavlju 3.1. objašnjena je komanda *db.createUser()* za kreiranje novog korisnika *myUserAdmin*. Sada se korišćenjem iste te komande kreira i korisnik readOnlyUser, što je prikazano na slici 7.

```
Enterprise admin> use peopleDB
switched to db peopleDB
Enterprise peopleDB> db.createUser({
...   user: "readOnlyUser",
...   pwd: "rou123",
...   roles: [
...     { role: "read", db: "peopleDB" }
...   ]
... })
{ ok: 1 }
Enterprise peopleDB>
```

Slika 7. Kreiranje readOnly korisnika

Kao što je ranije pomenuto, uloga *read* obezbeđuje pristup za čitanje dodeljivanjem sledećih radnji: changeStream, collStats, dbHash, dbStats, find, killCursors, listCollections, listIndexes, listSearchIndexes.

Na slici 8 može se videti upotreba komande *db.getCollectionNames()* koja vraća imena kolekcija u toj bazi podataka, u ovom slučaju to je *peopletDB* baza podataka, u kojoj se nalazi samo jedna kolekcija *people*.

```
Enterprise peopleDB> db.getCollectionNames()
[ 'people' ]
```

Slika 8. Prikaz rezultata izvršenja komande db.getCollectionNames()

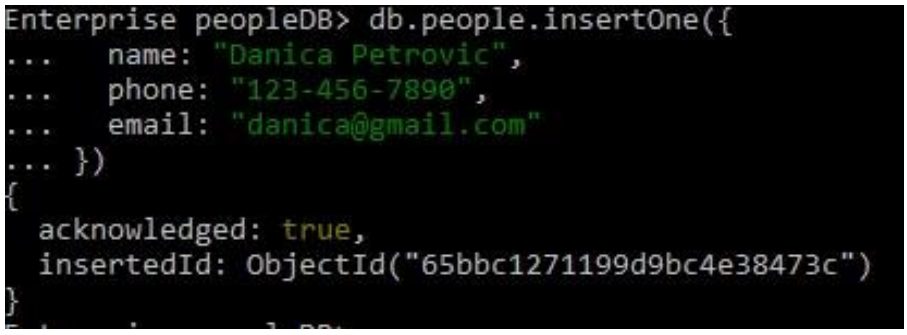
Na slici 9 može se videti upotreba komande *db.getCollectionInfos()* koja pruža informacije o kolekcijama u trenutnoj bazi podataka, u ovom slučaju to je *peopleDB* baza podataka.

Slika 9. Prikaz rezultata izvršenja komande db.getCollectionInfos()

3.2.2. Kreiranje readWrite korisnika i demonstracija njegovih privilegija

Uloga *readWrite* pruža sve privilegije uloge *read* plus mogućnost izmene podataka o svim nesistemskim kolekcijama. Uloga obezbeđuje sledeće radnje na tim kolekcijama: changeStream, collStats, convertToCapped, createCollection, createIndex, createSearchIndexes, dbHash, dbStats, dropCollection, dropIndex, dropSearchIndex, find, insert, killCursors, listCollections, listIndexes, listSearchIndexes, remove, renameCollectionSameDB, update, updateSearchIndex.

Na slici 10. Prikazan je postupak dodavanja novog podatka u bazu podataka *peopleDB* u kolekciju *people*, nad kojom korisnik *readWriteUser* i ima privilegiju upisa.



Slika 10. Prikaz rezultata izvršenja instert komande

Ako se sada pogleda sadržaj kolekcije *people* u bazi *peopleDB,* komandom *db.people.find().pretty()* može se videti da je dodat novi podatak, što je prikazano na slici 10.

```
Enterprise peopleDB> db.people.find().pretty()
[
  {
    _id: ObjectId("65bbc1271199d9bc4e38473c"),
    name: 'Danica Petrovic',
    phone: '123-456-7890',
    email: 'danica@gmail.com'
  }
]
```

Slika 11. Novi podatak u bazi podataka

3.3. Revizija

Da bi se izvršila konfiguracija revizije (auditing) u MongoDB-u, prvo je potrebno pronaći i otvoriti konfiguracioni fajl MongoDB servera, koji se obično naziva ***mongod.cfg***. U tom fajlu, u sekciji koja se odnosi na audit log, treba dodati ili prilagoditi određene parametre kako bi se omogućila revizija. Na slici 12 prikazane su izmene koje je potrebno uraditi u mongod.cfg fajlu kako bi se omogućila revizija.

```
#auditLog:
auditLog:
  destination: file
  format: JSON
  path: C:\Program Files\MongoDB\Server\7.0\log\auditLog.json
  filter: '{ atype: { $in: [ "createCollection", "dropCollection" ] } }'
```

Slika 12. Konfiguracija mongod.cfg fajla

Jedan od ključnih parametara je *destination*, koji određuje gde će se zapisivati rezultat revizije. Ukoliko želimo da se logovi čuvaju u fajlu, *destination* treba postaviti na vrednost *file*. Nakon toga, se koristi parametar *path* kako bi se specificirala putanja do fajla u koji će se zapisivati rezultat revizije. Zatim je potrebno postaviti parametar *format* koji određuje format u kojem će se logovi čuvati. Na primer, ako želimo koristiti JSON format, potrebno je postaviti parametar *format* na vrednost *JSON*. Filteri su takođe bitan deo konfiguracije jer omogućavaju da se odrede koji događaji će se pratiti. Ovi filteri uključuju opcije kao što su *dropCollection*, *createCollection* i *insert* omogućavajući da se definišu operacije koje će se pratiti.

Kada se završi sa unosom ovih parametara, potrebno je sačuvati promene u konfiguracionom fajlu. Da bi se primenile ove promene, potrebno je ponovo pokrenuti MongoDB server.

Primer konfiguracije koji je prikazan na slici 12 ima za cilj da omogući reviziju, beležeći logove u JSON formatu na odabranoj putanji i filtrirajući događaje prema zadatim kriterijumima.

Ukoliko se izvrši operacija db.createCollection("people”), kreira se log kao na slici 13. U JSON fajlu za reviziju, parametar *atype* označava tip operacije (npr. createCollection), parametar *ts* predstavlja tačno vreme izvršenja operacije. *Uuid* je jedinstveni identifikator poruke, a *local* sadrži informacije o MongoDB instanci na kojoj je operacija izvršena (IP adresa, port). U sekciji *remote* su informacije o klijentskoj konekciji koja je izazvala događaj (IP adresa, port). *Users* pruža detalje o korisniku koji je izvršio operaciju. *Roles* sadrži informacije o dodeljenim ulogama korisnika. *Params* nudi specifične detalje o operaciji (npr. ime kreirane kolekcije). *Result* beleži status uspešnosti operacije, gde 0 označava uspešno izvršenu operaciju. Ovi parametri omogućavaju kompletno praćenje i analizu svake revizionirane operacije.


```
{
   "atype": "createCollection",
   "ts": {
      "$date": "2024-01-25T18:24:20.447+0100"
   },
   "uuid": "123e4567-e89b-12d3-a456-426614174001",
   "local": {
      "ip": "127.0.0.1",
      "port": 27017
   },
   "remote": {
      "ip": "127.0.0.1",
      "port": 58157
   },
   "users": [
      {
         "user": "jelenadjikic",
         "db": "testDB"
      }
   ],
   "roles": [
      "admin"
   ],
   "params": {
      "collection": "people"
   },
   "result": 0
}
```


Slika 13. Primer audit loga za operaciju createCollection

3.4. CSFLE Enkripcija

CSFLE (Client-Side Field Level Encryption) enkripcija je tehnika enkripcije koja omogućava enkripciju određenih polja unutar dokumenata baze podataka direktno na strani klijenta. U ovom poglavlju urađen je celokupan primer konfiguracije i testiranja ove enkripcije, u node.js jednostavnoj aplikaciji.

3.4.1. Kreiranje master ključa (CMS)

Prvo je neophodno generisati ključ korisnika (Customer Master Key - CMS) koristeći Node.js modul *crypto* kako bi stvorio nasumični niz bajtova dužine 96. Ovaj ključ korisnika se čuva u datoteci pod nazivom *master-key.txt*.

Zatim je potrebno pročitati ključ korisnika iz datoteke i postaviti postavke pružaoca usluge za upravljanje ključevima (KMS - Key Management Service). U ovom slučaju, pružalac usluge je lokalni pružalac, a ključ korisnika se postavlja kao ključ za ovog lokalnog pružaoca usluge.

Na slici 14 prikazani su navedeni koraci koji omogućavaju generisanje i upotrebu ključa korisnika za enkripciju i dekripciju podataka.

```
// 1. Kreiranje Customer Master Key-a (CMS)
const fs = require("fs");
const crypto = require("crypto");
try {
  fs.writeFileSync("master-key.txt", crypto.randomBytes(96));
} catch (err) {
  console.error(err);
}

const provider = "local";
const path = "./master-key.txt";
const localMasterKey = fs.readFileSync(path);
const kmsProviders = {
  local: {
    key: localMasterKey,
  },
};
```

Slika 14. Kreiranje master ključa

3.4.2. Kreiranje jedinstvenog indeksa na kolekciji Key Vault

Prvo je neophodno povezati sa bazom podataka (MongoDB serverom), a zatim obrisati kolekciju za čuvanje ključeva *Key Vault* kako bi se osiguralo čišćenje prethodnih podataka i ključeva, što je prikazano na slici 15.

Nakon toga, kreira se delimični jedinstveni indeks na polju *keyAltNames* u kolekciji *Key Vault*. Ovaj indeks je deo mehanizma koji omogućava MongoDB serveru da obezbedi jedinstvenost alternativnih imena ključeva, što je važno za enkripciju na nivou polja na klijentskoj strani. Indeks je postavljen kao delimičan sa uslovom *partialFilterExpression* koji osigurava jedinstvenost samo za dokumenta gde *keyAltNames* postoji, što je bitno za ispravno funkcionisanje enkripcije na nivou polja.

```
const keyVaultDatabase = "encryption";
const keyVaultCollection = "__keyVault";
const keyVaultNamespace = `${keyVaultDatabase}.${keyVaultCollection}`;

const keyVaultClient = new MongoClient(uri);
await keyVaultClient.connect();

const keyVaultDB = keyVaultClient.db(keyVaultDatabase);
await keyVaultDB.dropDatabase();
await keyVaultClient.db("studentDB").dropDatabase();

// 2. Kreiranje jedinstvenog indeksa na kolekciji Key Vault
const keyVaultColl = keyVaultDB.collection(keyVaultCollection);
await keyVaultColl.createIndex(
  { keyAltNames: 1 },
  {
    unique: true,
    partialFilterExpression: { keyAltNames: { $exists: true } },
  }
);
```

Slika 15. Kreiranje indeksa na kolekciji Key Vault

3.4.3. Kreiranje ključa za enkripciju polja

Da bi se uspešno kreirao Data Encryption Key (DEK), korisnik baze podataka koji aplikacija koristi za povezivanje sa MongoDB mora imati dozvole *dbAdmin* na namespace-vima *encryption.__keyVault* i *studentDB* bazi podataka. Nakon uspostavljanja veze, potrebno je kreirati instancu *ClientEncryption* koristeći konkretnu klijentsku vezu, gde se navodi namespace za *Key Vault* kolekciju i KMS provider.

Zatim, koristeći instancu *ClientEncryption*, može se generisati DEK pozivom metode *createDataKey,* kao na slici 16 . Prilikom kreiranja DEK-a, potrebno je specificirati provajdera ključa i alternativna imena ključeva *keyAltNames*. U ovom slučaju, kreira se ključ sa alternativnim imenom *jmbgKey*.

Nakon izvršenja ovog dela koda, dobija se informacija o *DataKeyId*-u u obliku base64 stringa, što omogućava dalje upravljanje i praćenje DEK-a u okviru sistema enkripcije na strani klijenta.

```
// 3. Kreiranje ključa za enkripciju polja
const encryption = new ClientEncryption(client, {
  keyVaultNamespace,
  kmsProviders,
});

const key = await encryption.createDataKey(provider, {
  keyAltNames: ["jmbgKey"],
});
console.log("DataKeyId [base64]: ", key.toString("base64"));
```

Slika 16. Kreiranje ključa za enkripciju polja

3.4.4. Kreiranje šeme enkripcije

Postupak kreiranja šeme enkripcije počinje definisanjem ključa za enkripciju *dataKey*, koji je neophodan za proces enkripcije. Ovaj ključ je generisan kodom sa slike 23.

Kreiranje šeme enkripcije u MongoDB-u predstavlja ključni korak u obezbeđivanju sigurnosti podataka. Prvo svojstvo šeme enkripcije je *bsonType*, koje omogućava da se precizno odredi tip objekta koji će se enkriptovati. Nakon toga, svojstvo *encryptMetadata*, definiše metapodatke. To uključuje ID ključa za enkripciju.

Kroz svojstvo *properties*, precizira se kako će svako pojedinačno svojstvo objekta biti enkriptovano. Na ovom primeru enkriptuje se svojstvo *jmbg*, kome je tip string, a enkripcioni algoritam koji se koristi je "AEAD_AES_256_CBC_HMAC_SHA_512-Random", što je prikazano na slici 17.

```
// 4. Kreiranje šeme enkripcije
dataKey = "suegXPFkSjKAVp2VrVkuOg==";
const schema = {
  bsonType: "object",
  encryptMetadata: {
    keyId: [new Binary(Buffer.from(dataKey, "base64"), 4)],
  },
  properties: {
    jmbg: {
      encrypt: {
        bsonType: "string",
        algorithm: "AEAD_AES_256_CBC_HMAC_SHA_512-Random",
      },
    },
  },
};

var studentSchema = {};
studentSchema[namespace] = schema;
```

Slika 17. Kreiranje šeme enkripcije

3.4.5. Dodavanje novog dokumenta sa enkrptovanim poljem u bazu

Pre dodavanja novog podatka u bazu, potrebno je inicijalizovati MongoDB klijente, pri čemu se koristi jedan za sigurnu (enkriptovanu) komunikaciju s bazom podataka, a drugi za običnu (neenkriptovanu) komunikaciju, kao na slici 18.

```
const secureClient = new MongoClient(connectionString, {
  autoEncryption: {
    keyVaultNamespace,
    kmsProviders,
    schemaMap: studentSchema,
    extraOptions: extraOptions
  }
});
const regularClient = new MongoClient(connectionString);
```

Slika 18. Kreiranje sigurnog i običnog klijenta

secureClient namenjen je za sigurnu komunikaciju kod koga se osim stringa za konekciju dodaju i parametri za automatsko šifrovanje podataka u okviru *autoEncryption* objekta. *keyVaultNamespace* definiše gde će biti smešteni ključevi za enkripciju, a *kmsProviders* predstavlja pružaoca usluge za upravljanje ključevima. Takođe, koristii se *schemaMap* da se

definiše šema enkripcije za određene kolekcije, u ovom slučaju, studentSchema za kolekciju sa informacijama o studentima. Dodatne opcije, poput putanje do deljenje biblioteke, specificirane su kroz extraOptions.

Sa *regularClient-om* kreira se drugi MongoDB klijent za običnu, neenkriptovanu komunikaciju. Ovaj klijent se koristi za demonstraciju pregleda upisanih enkirptovanih podataka.

Nakon kreiranja potrebnih klijenata i njihovog konektovanja na bazu, preostalo je samo upisati novi dokument kod kojeg će biti enkritpovano polje *jmbg*, korišćenjem secure klijenta, kao na slici 19.

```
const writeResult = await secureClient
        .db(db)
        .collection(coll)
        .insertOne({
          name: 'Nikola Misic',
          jmbg: "1108996856954",
          phone: '123-456-7890',
          email: 'nikolamisic@gmail.com'
        });
```

Slika 19. Dodavanje novog podatka

Sada je dokument upisan u kolekciju *students* koja se nalazi u bazi *studentDB*. Ukoliko upit za listanje dokumenata izvrši secure klijent dobiće dokument kod kojeg je polje *jmbg* dekriptovano nazad u prvobitno stanje. Međutim ukoliko isti upit izvrši običan klijent dobiće enkriptovano *jmbg* polje i neće moći da ga pročita. Na slici 20 prikazan je kod izvršenja opisanih upita, a na slici 21 prikazan je rezulat koji dobijaju, redom, običan i secure klijent.

```
console.log("Finding a document with regular (non-encrypted) client. ");
console.log( await regularClient.db(db).collection(coll).findOne({ name: /Nikola Misic/ }));
console.log("Finding a document with encrypted client. ");
console.log( await secureClient.db(db).collection(coll).findOne({ name: /Nikola Misic/ }));
```

Slika 20. Izvršenje upita za pretragu kolekcije students

```
Finding a document with regular (non-encrypted) client.
{
  _id: new ObjectId('65bd4e4c40021be9539ecd7f'),
  name: 'Nikola Misic',
  jmbg: Binary.createFromBase64('ArLnoFzxZEoygFadla1ZLjoC6biBTE/BJGrPGj+S/5H/UaRYFvCjrlV1U7bSTw21p20PmcGIqOlfXOu0JMwfsvqk3cg
H/h+DJfySMpQjBkp0YTW5+EuMHSIxetnqx6kqmoo=', 6),
  phone: '123-456-7890',
  email: 'nikolamisic@gmail.com'
}
Finding a document with encrypted client.
{
  _id: new ObjectId('65bd4e4c40021be9539ecd7f'),
  name: 'Nikola Misic',
  jmbg: '1108996856954',
  phone: '123-456-7890',
  email: 'nikolamisic@gmail.com'
}
```

Slika 21. Poređenje rezultata koji dobijaju secure i običan klijent

4. Zaključak

Ovaj rad pruža uvid u bezbednosne aspekte sistema MongoDB, istražujući ključne elemente kao što su autentifikacija, autorizacija, revizija podataka i enkripcija. Analiza bezbednosne arhitekture MongoDB skladišta podataka ukazuje na visok nivo zaštite koji pruža organizacijama u savremenom okruženju aplikacija.

Autentifikacija, kroz različite mehanizme kao što su SCRAM, x.509 sertifikati, LDAP proxy i Kerberos, predstavlja osnovu sigurnog pristupa podacima. Autorizacija se postiže putem ugrađenih uloga, omogućavajući organizacijama precizno upravljanje privilegijama korisnika. Demonstracija rada sa različitim privilegijama pruža konkretan prikaz fleksibilnosti sistema u prilagođavanju specifičnim bezbednosnim zahtevima.

Revizija podataka omogućava praćenje promena, dok enkripcija podataka obezbeđuje kompletnu sigurnost tokom prenosa, mirovanja i upotrebe podataka. Ove bezbednosne funkcionalnosti čine MongoDB snažnim izborom za organizacije koje zahtevaju visok nivo zaštite podataka.

MongoDB se ističe kao robustna i skalabilna platforma sa visokim standardima bezbednosti, prilagođena savremenim zahtevima aplikacija. Ipak, adekvatno upravljanje izazovima i pažljivo vođenje računa o manama neophodno je kako bi se osigurala potpuna bezbednost sistema.

5. Literatura

[1] https://www.mongodb.com/collateral/mongodb-security-architecture
[2] https://itzone.com.vn/en/article/major-security-alert-as-40000-mongodb-databases-left-unsecured-on-the-internet-463/
[3] https://www.mongodb.com/docs/manual/core/authentication/
[4] https://www.mongodb.com/docs/manual/core/security-scram/#std-label-authentication-scram
[5] https://www.mongodb.com/docs/manual/core/security-x.509/#std-label-security-auth-x509
[6] https://www.mongodb.com/docs/manual/core/security-ldap/#std-label-security-ldap
[7] https://www.mongodb.com/docs/manual/core/kerberos/
[8] https://www.mongodb.com/docs/manual/core/authorization/
[9] https://www.mongodb.com/docs/manual/reference/built-in-roles/
[10] https://www.mongodb.com/docs/manual/core/auditing/
[11] https://www.prisma.io/dataguide/mongodb/mongodb-encryption
[12] https://www.mongodb.com/docs/manual/core/security-transport-encryption/
[13] https://www.mongodb.com/docs/manual/core/security-encryption-at-rest/
[14] https://www.mongodb.com/products/capabilities/security/encryption
[15] https://www.mongodb.com/docs/v7.0/core/csfle/
[16] https://www.mongodb.com/docs/v7.0/core/queryable-encryption/
[17] https://www.mongodb.com/docs/manual/core/security-users/
[18] https://www.mongodb.com/docs/manual/tutorial/create-users/
[19] https://www.mongodb.com/docs/manual/tutorial/configure-scram-client-authentication/#std-label-create-user-admin
[20] https://www.mongodb.com/docs/manual/core/csfle/quick-start/