MongoDB-де аутентификация және авторизация

Кіріспе

MongoDB өндірістік жүйелерде веб-қосымшалардың, аналитикалық сервистердің және деректерді өңдеу пайплайндарының негізгі сақтау қабаты ретінде жиі қолданылады. Сондықтан дерекқорға қолжетімділік дұрыс басқарылмаса, жүйедегі ең құнды актив – деректер – тәуекелге түседі. Қауіпсіздік тек “құпиясөз қоюмен” шектелмейді: кім қосыла алады, қандай дерекқорға кіре алады, қандай операция орындай алады – осының бәрі нақты ережелермен басқарылуы керек. MongoDB қауіпсіздігінің өзегі екі ұғымға тіреледі: аутентификация (кім екенін анықтау) және авторизация (не істеуге құқығы барын анықтау).

Негізгі ұғымдар: аутентификация vs авторизация

Аутентификация - жүйеге қосылған субъектінің (адам, сервис, қосымша) кім екенін дәлелдеу процесі. Практикада бұл пайдаланушы аты мен құпиясөз, сертификат немесе сыртқы провайдер арқылы орындалуы мүмкін. MongoDB-де аутентификация қосылмаса, дерекқорға қосылған кез келген клиент өзін “кім” екенін дәлелдемей-ақ әрекет ете алады, бұл өндірістік орта үшін қауіпті.

Авторизация - аутентификациядан өткен пайдаланушыға қандай әрекеттерге рұқсат берілетінін анықтайды. Мысалы, қолданбаға тек оқу/жазу керек болуы мүмкін, ал әкімшіге пайдаланушыларды басқару қажет. Авторизация дұрыс қойылмаса, “артық құқық” мәселесі пайда болады: бір аккаунт бұзылса, бүкіл дерекқор толық бақылауға өтеді.

MongoDB қауіпсіздік моделі: пайдаланушы, рөл, артықшылықтар

MongoDB-де құқықтар рөлдер арқылы беріледі. Рөл – белгілі бір әрекеттер жиынтығына рұқсат (привилегиялар) беретін “пакет”. Пайдаланушыға бір немесе бірнеше рөл тағайындалады. Осылайша жүйе “пайдаланушы не істей алады?” деген сұраққа оның рөлі арқылы жауап береді.

Практикалық тұрғыда рөлдер екіге бөлінеді:

1. Built-in рөлдер – MongoDB ұсынатын дайын рөлдер (мысалы, оқу, оқу+жазу, әкімшілік рөлдер).
2. Custom рөлдер – нақты ұйым/жоба талаптарына сай жасалатын арнайы рөлдер (мысалы, тек белгілі коллекцияларға ғана жазуға рұқсат).

Бұл тәсілдің негізгі артықшылығы – басқарудың қарапайымдылығы: көптеген пайдаланушыға бірдей саясат қолдану үшін оларға бір рөл тағайындау жеткілікті.

Аутентификация қалай іске асады?

MongoDB-ге клиент қосылған кезде, сервер оның өзін кім екенін тексереді. Аутентификация механизмі конфигурацияға тәуелді. Жалпы логика мынадай:

- пайдаланушы дерекқорда (көбіне admin немесе нақты қолданба базасында) тіркелген болады;
- клиент қосылғанда өзінің идентификаторын және құпия ақпаратын ұсынады;
- MongoDB бұл деректерді тексеріп, сәтті болса сессияға рұқсат береді.

Өндірісте ең басты талап – аутентификацияны міндетті түрде қосу. Себебі аутентификациясыз MongoDB “қорғалған жүйе” бола алмайды, тіпті веб-қосымшаңызда авторизация бар болса да. Дерекқорға тікелей қол жеткізген адам қолданбадағы шектеулерді айналып өтуі мүмкін

Авторизация қалай жұмыс істейді?

Аутентификациядан өткен соң, MongoDB пайдаланушының рөлдерін қарап, нақты әрекетке рұқсат бар-жоғын тексереді. Мысалы, қолданба orders коллекциясына жазба қосуды сұраса, сервер пайдаланушының рөлінде сол операцияға (insert) және сол ресурсқа (нақты дерекқор/коллекция) рұқсат бар-жоғын тексереді. Егер рұқсат жоқ болса, операция орындалмайды.

“Минималды құқық” қағидасы (Principle of Least Privilege)

Қауіпсіз жүйені жобалауда негізгі қағида – минималды қажетті құқық. Яғни пайдаланушыға (немесе сервиске) тек нақты қажет құқықтарды ғана беру керек, артық құқық берілмеуі тиіс.

Мысалы:

- веб-қосымшаға әкімші құқықтың қажеті жоқ; оған көбіне белгілі базадағы кей коллекцияларға оқу/жазу жеткілікті;
- аналитика сервисіне тек оқу құқығы жеткілікті болуы мүмкін;
- әкімшілік аккаунттар саны аз болуы керек және олар тек әкімшілік жұмыста қолданылуы тиіс.

Бұл қағида практикалық қауіптерді күрт азайтады: егер қолданба аккаунты “ағып кетсе”, шабуылдаушы бүкіл серверді басқара алмайды, тек сол аккаунттың шектеулі рұқсатымен ғана әрекет етеді.

Рөлдерге негізделген практикалық сценарий

“Университет жүйесін” алайық:

- **appUser** (қосымша аккаунты): студенттер, пәндер, бағалар коллекцияларына оқу/жазу, бірақ пайдаланушыларды басқаруға рұқсат жоқ;
- **analystUser**: тек оқу құқығы, себебі ол есептер шығарады;
- **adminUser**: пайдаланушыларды жасау/өшіру, рөлдер тағайындау, қауіпсіздік параметрлерін басқару.

Жиі кездесетін қателер және олардың салдары

MongoDB қауіпсіздігінде жиі қайталанатын қателердің бірнешеуі бар.

- Бірінші қате – аутентификацияны қоспай, дерекқорды желіге ашық қалдыру. Мұндай жағдайда дерекқорға кез келген клиент қосыла алады.
- Екінші қате – қолданбаға admin сияқты артық құқық беру. Бұл “бәрі жұмыс істесін” деген мақсатпен жасалады, бірақ қауіпсіздік тұрғысынан ең қауіпті шешім: қолданба аккаунты бұзылса, бүкіл дерекқор толық бақылауға өтеді.
- Үшінші қате – құпия ақпаратты (connection string, пароль) кодта немесе ашық репозиторийде сақтау. Бұл жағдайда парольдің “ағып кетуі” өте оңай.
- Төртінші қате – бірдей парольдерді ұзақ қолдану және әкімшілік аккаунттарды күнделікті жұмысқа пайдалану. Бұл да тәуекелді арттырады

Қорытынды

MongoDB-де қауіпсіздіктің негізгі тіректері – аутентификация және авторизация. Аутентификация “кім қосылды?” деген сұраққа жауап береді, ал авторизация “ол не істей алады?” дегенді рөлдер арқылы басқарады. Өндірістік жүйеде аутентификацияны міндетті түрде қосып, құқықтарды минималды қажетті деңгейде бөлу керек. Дұрыс ұйымдастырылған рөлдер мен пайдаланушылар жүйені қауіпсіз және басқаруға ыңғайлы етеді.

Деректерді шифрлау

Кіріспе

Қауіпсіздік тек пайдаланушы/рөлдермен шектелмейді. Тіпті авторизация дұрыс болса да, деректердің “үшінші жолмен” қолға түсіп қалу қаупі бар: дискті ұрлап кету, резервтік көшірменің сыртқа шығуы, желі трафигін тыңдау, бұлттағы конфигурация қателігі, немесе ішкі жүйелер арасындағы байланыс бұзылуы. Осындай сценарийлерде деректерді қорғаудың ең сенімді тәсілдерінің бірі – шифрлау. Шифрлау екі негізгі жерде қолданылады: сақтауда (at rest) және берілісте (in transit). Біріншісі – деректер дискіде жатқанда қорғалады, екіншісі – деректер желі арқылы өткен кезде қорғалады.

Негізгі ұғымдар: “at rest” және “in transit”

At rest (сақтауда) – деректер файл жүйесінде, дискіде, snapshot-та, backup-та, немесе дерекқордың сақтау қабатында тұрғанда қорғалады. Мұнда қауіп: біреу дискіге немесе резервтік көшірмеге қол жеткізсе, деректерді оқып қоюы мүмкін.

In transit (берілісте) – клиент (қосымша) пен MongoDB сервері арасында немесе сервистер арасында деректер желімен жүргенде қорғалады. Мұнда қауіп: трафикті тыңдау (sniffing), “аралықтағы адам” шабуылы (MITM), немесе корпоративтік желіде дұрыс бөлінбеген сегмент арқылы дерек ағып кетуі мүмкін.

Екеуі де маңызды, себебі қауіптер әртүрлі: біріншісі физикалық/файлдық қолжетімділікпен байланысты, екіншісі – желілік байланыспен байланысты.

Сақтаудағы шифрлау (Encryption at Rest): не қорғалады?

At rest шифрлау деректердің келесі түрлерін қорғай алады:

- дерекқор файлдары (data files);
- индекстер;
- журналдар (логтар);
- snapshot/backup файлдары;
- кейбір ортада swap немесе уақытша файлдар.

Нәтижесінде, біреу дискіге тікелей қол жеткізгенімен (мысалы, серверді алып кетсе немесе бұлтта диск snapshot-ы сыртқа шықса), деректер “ашық мәтін” ретінде оқылмайды. Яғни шифрлау деректерді “қолға түссе де оқылмайтындай” етеді.

MongoDB-де at rest шифрлау тәсілдері

MongoDB-мен жұмыс істейтін өндірістік жүйелерде at rest шифрлау екі деңгейде кездеседі:

- Бірінші деңгей – инфрақұрылым деңгейі: диск/том шифрланады (операциялық жүйе немесе бұлттық провайдер арқылы). Бұл тәсіл көбіне ең қолжетімді және кең таралған, себебі дерекқорға тәуелсіз жұмыс істейді.
- Екінші деңгей – дерекқор деңгейі: дерекқордың өзі сақтау қабатында шифрлауды қолдауы мүмкін (кей ортада арнайы мүмкіндіктермен). Бұл тәсіл дерекқорға жақын, басқаруы нақтырақ болуы мүмкін, бірақ конфигурациясы күрделірек болуы ықтимал.

Кілттерді басқару (Key Management): ең маңызды бөлік

Шифрлау күшті болуы үшін ең шешуші фактор – кілттің қауіпсіздігі. Егер кілт қолда болса, шифрлау да мәнін жоғалтады.

Практикада “кілттерді басқару” мыналарды қамтиды:

- кілттерді кодта/репозиторийде сақтамау;
- кілттерді қауіпсіз қоймада сақтау (мысалы, арнайы secret manager);
- кілттерді ауыстыру саясаты (rotation);
- қолжетімділікті шектеу (кім көре алады, кім пайдалана алады);
- аудит (кілтке кім қол жеткізді).

Кілттерді басқару (Key Management): ең маңызды бөлік

Шифрлау күшті болуы үшін ең шешуші фактор – кілттің қауіпсіздігі. Егер кілт қолда болса, шифрлау да мәнін жоғалтады.

Практикада “кілттерді басқару” мыналарды қамтиды:

- кілттерді кодта/репозиторийде сақтамау;
- кілттерді қауіпсіз қоймада сақтау (мысалы, арнайы secret manager);
- кілттерді ауыстыру саясаты (rotation);
- қолжетімділікті шектеу (кім көре алады, кім пайдалана алады);
- аудит (кілтке кім қол жеткізді).

Берілістегі шифрлау (Encryption in Transit): TLS/SSL идеясы

In transit шифрлау – деректер желімен жүргенде шифрланған күйде берілуі. Бұл әдетте TLS (SSL деп те аталады) арқылы іске асады.

TLS не береді?

- трафикті тыңдаған адам деректі оқи алмайды;
- сервердің түпнұсқалығын тексеруге мүмкіндік береді (сертификат арқылы);
- байланысқа өзгеріс енгізуді (MITM) қиындатады.

MongoDB клиенттері мен драйверлері (қосымша) әдетте TLS арқылы қосылуды қолдайды. Өндірісте сыртқы желі арқылы қосылсаңыз, TLS – міндетті талап.

TLS қолданбағандағы нақты тәуекелдер

TLS болмаса, желідегі деректер ашық күйде жүруі мүмкін. Бұл келесі қауіптерді тудырады:

- логин/пароль немесе токендер ұсталып қалуы
- сұраныстағы жеке деректер (аты-жөні, телефон, баға, транзакция) оқылып қалуы
- аралықтағы адам шабуылы арқылы дерек өзгертілуі (теориялық және кей желілерде практикалық)

Әсіресе қоғамдық Wi-Fi, дұрыс сегменттелмеген корпоративтік желі немесе бұлттағы қауіпсіздік тобы қате қойылған ортада TLS қолданбау өте қауіпті.

TLS қолданбағандағы нақты тәуекелдер

TLS болмаса, желідегі деректер ашық күйде жүруі мүмкін. Бұл келесі қауіптерді тудырады:

- логин/пароль немесе токендер ұсталып қалуы
- сұраныстағы жеке деректер (аты-жөні, телефон, баға, транзакция) оқылып қалуы
- аралықтағы адам шабуылы арқылы дерек өзгертілуі (теориялық және кей желілерде практикалық)

Әсіресе қоғамдық Wi-Fi, дұрыс сегменттелмеген корпоративтік желі немесе бұлттағы қауіпсіздік тобы қате қойылған ортада TLS қолданбау өте қауіпті.

Қорытынды

Шынайы өндірістік қауіпсіздік әдетте “бір құралмен” шешілмейді. At rest шифрлау дискі/backup қолға түссе де қорғайды. In transit шифрлау желі арқылы өтетін деректерді қорғайды. Екеуі бірігіп қолданылғанда, жүйе қауіптің бірнеше түріне төзімді болады: бір қабат бұзылса да, екінші қабат қорғаныс береді. Мұны “қорғаныстың қабатталуы” (defense in depth) деп түсіндіруге болады.

Деректерді шифрлау – өндірістік жүйелерде міндетті қауіпсіздік практикасы. At rest шифрлау деректерді дискіде және backup-та қорғайды, ал in transit шифрлау желідегі трафикті қорғайды. Ең маңыздысы – кілттерді басқару және backup қауіпсіздігін ұмытпау. MongoDB жүйесін нақты ортада қолданғанда, шифрлау рөлдер/пайдаланушылармен бірге толық қауіпсіздік архитектурасын құрайды.

Резервтік көшіру және деректерді қалпына келтіру

Кіріспе

Кез келген дерекқор жүйесінде ең қауіпті сұрақтардың бірі – “егер дерек жоғалса не істейміз?”. Деректер жоғалуы тек сервердің бұзылуынан болмайды. Көп жағдайда деректер адам қателігінен (қате өшіру, қате update), бағдарламалық қателіктен (bug), шабуылдан (ransomware), конфигурацияның бұзылуынан немесе апаттан (диск істен шығу, дата-орталық проблемасы) кетеді. Сондықтан резервтік көшіру – бұл “қосымша опция” емес, дерекқормен жұмыс істеудің міндетті өндірістік мәдениеті. Бірақ backup өз алдына жеткіліксіз: ең маңыздысы – қалпына келтірудің жұмыс істейтініне көз жеткізу, яғни restore процесін жоспарлау және тексеру.

Негізгі ұғымдар: RPO және RTO

Backup стратегиясын түсіндіруде екі өлшем маңызды:

RPO (Recovery Point Objective) – “қанша деректі жоғалтуға болады?” деген сұрақ. Мысалы, RPO = 15 минут болса, біз ең көбі соңғы 15 минуттық өзгерісті жоғалтуға келісеміз. Бұл backup жиілігімен байланысты.

RTO (Recovery Time Objective) – “қалпына келтіруге қанша уақыт кетуі мүмкін?” деген сұрақ. Мысалы, RTO = 1 сағат болса, жүйені бір сағат ішінде қайта тұрғызу керек. Бұл restore жылдамдығымен, инфрақұрылыммен, автоматтандырумен байланысты.

MongoDB-де backup-тың негізгі тәсілдері

Логикалық backup: mongodump. Бұл тәсіл деректерді BSON форматында шығарып, кейін қайта қалпына келтіруге мүмкіндік береді. Оқыту процесінде және орта көлемдегі жобаларда ең түсінікті жолдың бірі.

Бүкіл дерекқорды dump жасау:

```
mongodump --uri="mongodb://USER:PASS@HOST:27017/DBNAME" --out ./backup_$(date +%F)
```

Тек бір коллекцияны алу:

```
mongodump --uri="mongodb://USER:PASS@HOST:27017/DBNAME" \
  --collection students --out ./backup_students
```

Белгілі шартпен ғана backup (мысалы, 2026 жылғы логтар):

```
mongodump --uri="mongodb://USER:PASS@HOST:27017/DBNAME" \
  --collection logs \
  --query '{"year": 2026}' \
  --out ./backup_logs_2026
```

Компрессиямен archive жасау (көлемді азайтады):

```
mongodump --uri="mongodb://USER:PASS@HOST:27017/DBNAME" --archive=backup.archive --gzip
```

JSON экспорт/импорт: mongoexport / mongoimport (оқытуға ыңғайлы)

Бұл тәсіл көбіне **оқу/демонстрация** үшін қолайлы, себебі нәтиже JSON/CSV сияқты оқылатын форматта болады. Бірақ өндірістегі толық backup ретінде әрдайым тиімді емес (үлкен деректе баяу/ауыр болуы мүмкін).

Экспорт (JSON):

```
mongoexport --uri="mongodb://USER:PASS@HOST:27017/DBNAME" \
  --collection students --out students.json
```

Импорт:

```
mongoimport --uri="mongodb://USER:PASS@HOST:27017/DBNAME" \
  --collection students --file students.json
```

Физикалық snapshot (инфрақұрылым деңгейі)

Snapshot тәсілінде дерекқордың файлдары диск/том деңгейінде көшіріледі. Бұл көбіне жылдам болады, бірақ консистенттілік (бір сәттік күй) дұрыс сақталуы керек.

Репликация қолжетімділікті арттырады, бірақ **backup-тың орнын баспайды**. Себебі қате өшіру немесе зиянды өзгеріс репликаға да таралуы мүмкін. Сондықтан реплика бар болса да, бөлек резервтік көшірме қажет.

Backup-тың басты талабы – оның бірізді (consistent) болуы. Егер жүйе жұмыс істеп тұрғанда көшірме дұрыс алынбаса, backup ішінде деректердің бөлігі жаңа, бөлігі ескі болуы мүмкін. Мұндай backup қалпына келтірілгенде қателіктер шығады немесе деректер қисынсыз болады.

Қалпына келтіру: mongorestore

Папкадан толық қалпына келтіру:

```
mongorestore --uri="mongodb://USER:PASS@HOST:27017" ./backup_2026-02-14
```

Archive+gzip-тен қалпына келтіру:

```
mongorestore --uri="mongodb://USER:PASS@HOST:27017" --archive=backup.archive --gzip
```

Тек бір коллекцияны қалпына келтіру:

```
mongorestore --uri="mongodb://USER:PASS@HOST:27017" \
  --nsInclude="DBNAME.students" ./backup_students
```

Қалпына келтірерде “таза” көтеру үшін (drop):

```
mongorestore --uri="mongodb://USER:PASS@HOST:27017" --drop ./backup_2026-02-14
```

Қандай қалпына келтіру сценарийлері болады?

Қалпына келтіру әрдайым толық апат емес. Практикада:

- толық апатта бүкіл жүйе қайта көтеріледі;
- логикалық қате болса (қате delete/update) кейде бір коллекцияны немесе нақты уақыт нүктесін қайтару маңызды;
- қауіпсіздік инцидентінде “таза” күйге қайтару қажет болуы мүмкін.

Қорытынды

MongoDB-де резервтік көшіру және қалпына келтіру – өндірістік сенімділіктің негізі. Backup стратегиясы RPO және RTO талаптарына сүйеніп құрылады. Backup-тарды қауіпсіз сақтау, шифрлау, және ең бастысы – restore тестін жүйелі өткізу сенімді жүйе құруға көмектеседі. Ал практикада mongodump және mongorestore – ең негізгі құралдар болып саналады.

MongoDB-де транзакциялар және ACID қағидалары

Кез келген ақпараттық жүйеде деректердің дұрыстығы ең маңызды талаптардың бірі болып саналады. Әсіресе банктік жүйелерде, интернет-дүкендерде, университеттік платформаларда, төлем сервистерінде немесе аналитикалық жүйелерде бірнеше операция бір-бірімен тығыз байланысты болады. Егер сол операциялардың бір бөлігі ғана орындалып, қалғаны қате тоқтап қалса, жүйедегі деректер бұзылуы мүмкін. Осындай мәселелерді болдырмау үшін дерекқор жүйелерінде транзакциялар қолданылады.

Транзакция – бір логикалық әрекетті құрайтын операциялар жиынтығы. Бұл операциялар толық орындалуы тиіс немесе мүлде орындалмауы керек. Яғни жүйе “аралық” қате күйде қалмауы қажет.

ACID қағидалары дегеніміз не?

Транзакциялардың негізінде ACID қағидалары жатыр. Бұл – сенімді дерекқор жүйесінің негізгі төрт қасиеті.

ACID атауы төрт терминнің алғашқы әріптерінен құралған:

- Atomicity
- Consistency
- Isolation
- Durability

Осы қасиеттер транзакциялардың дұрыс және қауіпсіз орындалуын қамтамасыз етеді.

Atomicity – атомарлық

Atomicity қағидасының негізгі идеясы: транзакция толық орындалады немесе толық орындалмайды.

Мысалы, банк жүйесінде бір қолданушыдан екіншісіне ақша аудару процесін қарастырайық. Бұл жерде кемінде екі операция орындалады:

1. Бірінші шоттан ақша алу.
2. Екінші шотқа ақша қосу.

Егер бірінші операция орындалып, екіншісі қате тоқтап қалса, ақша “жоғалып кетеді”. Сондықтан екі операция бір бүтін транзакция ретінде қарастырылуы керек.

MongoDB-де транзакция ішінде орындалған барлық өзгерістер commit кезінде ғана тұрақты түрде сақталады. Егер қандай да бір қате болса, rollback жасалып, барлық өзгеріс жойылады.

Consistency – біріздділік

Consistency қағидасы транзакция орындалғаннан кейін дерекқор әрқашан дұрыс және рұқсат етілген күйде қалуы керек дегенді білдіреді.

Мысалы:

```
db.students.createIndex(
{ email: 1 },
{ unique: true }
)
```

Бұл жағдайда бірдей email екі рет енгізілмейді.

Consistency қағидасы деректердің “логикалық дұрыстығын” сақтайды.

Isolation – оқшаулау

Isolation дегеніміз бірнеше транзакция бір уақытта орындалса да, олар бір-біріне кедергі жасамауы керек.

Өндірістік жүйелерде жүздеген немесе мыңдаған қолданушы бір мезетте жұмыс істейді.

Мысалы, екі оператор бір тауардың соңғы данасын бір уақытта сатып жіберуі мүмкін. Егер isolation дұрыс болмаса, қоймада “минус” пайда болады.

MongoDB транзакциялар кезінде snapshot isolation принципін қолданады. Бұл дегеніміз транзакция белгілі бір уақыттағы деректердің тұрақты “көшірмесімен” жұмыс істейді.

Durability – тұрақтылық

Durability қағидасы commit орындалғаннан кейін деректер жоғалмауы керек дегенді білдіреді. Яғни, сервер өшсе де, жүйе қайта жүктелсе де, апат болса да, сақталған өзгерістер қалуы тиіс. MongoDB durability-ді бірнеше механизм арқылы қамтамасыз етеді:

- Journal – өзгерістерді алдымен арнайы журнал файлына жазу механизмі.
- Replication – деректердің бірнеше серверде сақталуы.
- Majority write concern – өзгеріс replica set мүшелерінің көпшілігіне жеткеннен кейін ғана commit деп есептеледі.

Мысалы:

```
db.orders.insertOne(
{ item: "Laptop", price: 500000 },
{ writeConcern: { w: "majority" } }
)
```

Бұл жағдайда операция replica set мүшелерінің көпшілігі растағаннан кейін ғана сәтті орындалды деп саналады.

MongoDB-де транзакциялар қалай жұмыс істейді?

MongoDB-де транзакциялар session арқылы орындалады.

Жалпы логика:

1. Session ашылады.
2. Transaction басталады.
3. Бірнеше операция орындалады.
4. Commit немесе rollback жасалады.

```
const session = db.getMongo().startSession()
session.startTransaction()
try {
  const students =
    session.getDatabase("university").students
  const statistics =
    session.getDatabase("university").statistics
  students.updateOne(
    { _id: 1 },
    { $set: { grade: 95 } }
  )   statistics.updateOne(
    { subject: "Math" },
    { $inc: { averageUpdated: 1 } }
  )   session.commitTransaction()
} catch (e) {
  session.abortTransaction() }
```

Транзакциялардың кемшіліктері және шектеулері

Транзакциялар деректердің тұтастығын сақтауға көмектескенімен, олар жүйеге қосымша жүктеме түсіреді. Әрбір транзакцияны орындау кезінде сервер қосымша жадты пайдаланады, операцияларды уақытша бұғаттайды және серверлер арасындағы синхрондауды жүзеге асырады. Соның нәтижесінде жүйенің жалпы өнімділігі төмендеуі мүмкін.

Әсіресе ұзақ уақыт орындалатын транзакциялар қауіпті болып саналады. Мұндай жағдайда:

- ресурстар ұзақ уақыт босатылмайды;
- басқа операциялардың күту уақыты артады;
- жадты пайдалану көлемі көбейеді;
- репликацияның кешігуі (replication delay) пайда болуы мүмкін.

Қорытынды

MongoDB-де транзакциялар ACID қағидалары арқылы деректердің сенімділігі мен тұтастығын қамтамасыз етеді. Atomicity операциялардың толық орындалуын бақылайды, Consistency деректердің дұрыс күйін сақтайды, Isolation бір уақытта орындалатын транзакцияларды оқшаулайды, ал Durability commit жасалған өзгерістердің жоғалмауына кепіл береді.

MongoDB мониторинг және профильдеу

Кіріспе

MongoDB жүйесі “бір рет қойып, ұмытып кететін” технология емес. Әсіресе өндірістік ортада дерек көлемі өседі, сұраныстар көбейеді, индекстер өзгеруі мүмкін, қолданушылар саны артады. Мұндай кезде жүйенің жағдайын тұрақты бақыламасаңыз, мәселе көбіне кеш байқалады: қолданба баяулайды, CPU 100% болады, диск толады, реплика артта қалады, немесе кенеттен қате көбейеді.

Мониторингтің мақсаты – проблеманы “болып кеткеннен кейін” емес, ертерек байқау және нақты қай жерде қиындық барын түсіну. Ал профильдеу – сол проблеманың “тамырын” табуға көмектеседі: нақты қай сұраныстар баяу, қандай операциялар көп уақыт алады, неге.

Мониторинг пен профильдеу айырмашылығы

Мониторинг дегеніміз – жүйенің денсаулығын (health) үздіксіз бақылау. Мұнда біз ресурстарды және негізгі метрикаларды қараймыз: CPU, RAM, диск I/O, желі, қосылымдар саны, операциялар қарқыны, репликация күйі, кешігулер.

Профильдеу – сұраныстар мен операцияларды терең талдау. Яғни “неге баяулады?” деген сұраққа жауап. Профильдеу әдетте баяу сұраныстарды табу, explain() көру, profiler жазбаларын талдау арқылы жасалады.

Жедел тексеру құралдары (mongosh): жалпы статистика

Сервер статусы (жалпы жағдай):

```
db.serverStatus()
```

Бұл команда өте үлкен ақпарат береді: операциялар, жад, байланыстар, желілер, метрикалар және т.б. Практикада біз оның бәрін жаттамаймыз, бірақ “қай бөлім қай нәрсеге жауап береді” деген түсінік керек: мысалы connections – қосылымдар, opcounters – операциялар саны, mem – жад, wiredTiger – storage/cache метрикалары.

Қосылымдар және «connection storm»

MongoDB-де қосылым саны кейде күтпеген жерден өсіп кетеді. Бұл көбіне қолданбада байланыс пул-дарын дұрыс қойылмаған кезде болады. Нәтижесінде MongoDB көп қосылымды ұстап тұрып, ресурс жоғалтады. Сондықтан мониторингте байланыстар метрикасы маңызды.

Қосылымдарды көру:

```
db.serverStatus().connections
```

Баяу сұраныстар: мониторингтегі ең пайдалы сигнал

Өнімділік мәселесі көбіне “баяу сұраныстар көбейді” дегеннен басталады. Егер жүйеде баяу сұраныстар артса, қолданбада күту уақыты өседі. Сондықтан мониторингте баяу сұраныс көрсеткіштерін бақылау және профильдеу арқылы нақты сұранысты табу маңызды.

Бұл жерде екі қадам бар:

1. Баяу сұраныстар бар екенін байқау (симптом).
2. Қай сұраныс баяу екенін табу (себеп).

Профильдеу: Database Profiler қосу

MongoDB-де Database Profiler белгілі шартқа сай операцияларды арнайы жүйелік коллекцияға жаза алады. Бұл баяу сұраныстарды табудың өте пайдалы жолы.

Профильдеу деңгейін орнату (мысалы, 100ms-тен баяу сұраныстар):

```
db.setProfilingLevel(1, { slowms: 100 })
```

Мұнда 1 дегеніміз – “тек баяу операцияларды жаз” деген режим. Ал slowms – шекті уақыт.

Profiler нәтижесін оқу

Profiler жазбалары әдетте system.profile коллекциясына түседі. Сол жерден соңғы баяу операцияларды қарап, нақты қандай сұраныста проблема екенін табуға болады.

Соңғы баяу сұраныстарды көру:

```
db.system.profile.find().sort({ ts: -1 }).limit(10)
```

Бұл нәтижеде операция түрі, орындалу уақыты, қандай коллекция, қандай фильтр болғаны сияқты маңызды ақпарат болады. Мұны көргеннен кейін келесі қадам – сол сұранысты explain() арқылы тексеру және индекс/схема/сұранысты түзету.

explain() – профилирлеудің “жалғасы”

Profiler сізге “қай сұраныс баяу?” дегенді көрсетеді, ал explain() “неге баяу?” дегенге жауап береді. Сондықтан практикада бұл екеуі бірге жүреді.

Мысал:

```
db.logs.find({ userId: ObjectId("...") }).sort({ createdAt: -1 }).explain("executionStats")
```

Қорытынды

MongoDB жүйесінде мониторинг пен профильдеу деректер қорының тұрақты, сенімді және өнімді жұмысын қамтамасыз етудің негізгі құралдары болып табылады. Мониторинг арқылы сервердің ресурстық жағдайы, қосылымдар саны, диск жүктемесі, репликация күйі және операциялардың орындалуы үздіксіз бақыланады. Бұл жүйедегі ақауларды ерте анықтап, олардың үлкен мәселеге айналуына жол бермейді.

Ал профильдеу нақты баяу сұраныстарды, тиімсіз операцияларды және өнімділікке әсер ететін факторларды табуға мүмкіндік береді. MongoDB Profiler және explain() механизмі арқылы сұраныстың қалай орындалатыны талданып, индекстердің дұрыс қолданылуы тексеріледі. Соның нәтижесінде жүйенің жұмыс жылдамдығын арттыруға және сервер жүктемесін азайтуға болады.

Қорытынды

MongoDB жүйесінде мониторинг пен профильдеу деректер қорының тұрақты, сенімді және өнімді жұмысын қамтамасыз етудің негізгі құралдары болып табылады. Мониторинг арқылы сервердің ресурстық жағдайы, қосылымдар саны, диск жүктемесі, репликация күйі және операциялардың орындалуы үздіксіз бақыланады. Бұл жүйедегі ақауларды ерте анықтап, олардың үлкен мәселеге айналуына жол бермейді.

Ал профильдеу нақты баяу сұраныстарды, тиімсіз операцияларды және өнімділікке әсер ететін факторларды табуға мүмкіндік береді. MongoDB Profiler және explain() механизмі арқылы сұраныстың қалай орындалатыны талданып, индекстердің дұрыс қолданылуы тексеріледі. Соның нәтижесінде жүйенің жұмыс жылдамдығын арттыруға және сервер жүктемесін азайтуға болады.

Big Data үшін MongoDB деректер схемасын жобалаудың үздік тәжірибелері

Кіріспе

Big Data жағдайында мәселе тек “деректі сақтай салу” емес. Деректер көлемі жылдам өседі, сұраныстар саны артады, жүйе ұзақ уақыт жұмыс істейді, ал талаптар өзгеріп отырады. MongoDB схемаға бағытталған болғанымен, Big Data жүйесінде “схема жоқ” деп ойлау қауіпті. Сондықтан Big Data үшін схема жобалау – бұл тек дерек құрылымы емес, ол **өсу стратегиясы**, **сұраныс стратегиясы**, **индекс стратегиясы**, және **масштабтау стратегиясының** біріктірілген шешімі.

“Query-first” қағидасы: схеманы сұраныстарға қарап құру

MongoDB-де ең дұрыс модельдеу тәсілі – алдымен жүйедегі негізгі сұраныстарды (query patterns) анықтау. Big Data жүйесінде “қандай экрандар жиі ашылады?”, “қандай есептер көп құрылады?”, “қандай фильтр/сорт тұрақты қолданылады?” деген сұрақтар шешуші. Егер сұраныс белгісіз болса, схема кездейсоқ қалыптасады: бір жерде embedded, бір жерде references, индекстер ретсіз болады. Ал query-first қағидасында біз алдымен 5-10 негізгі сұранысты бекітеміз, содан кейін деректер сол сұраныстарды тез орындауға бейімделеді. Бұл Big Data-да өте маңызды, себебі үлкен көлемде кез келген қателік қымбатқа түседі.

Өсу заңдылығы: “қандай дерек шексіз өседі?”

Big Data жүйесінде ең үлкен қауіп – өсуі шексіз болатын деректерді дұрыс басқармау. Әдетте шексіз өсетіндер: логтар, оқиғалар (events), транзакциялар, пайдаланушы әрекеттері, сенсор өлшемдері, қатысу/баға тарихы. Бұларды бір құжаттың ішіне массив ретінде жинау – классикалық антипаттерн, себебі құжат семіреді, операция қымбаттайды, ақырында 16MB лимитіне тіреледі. Сондықтан ең үздік тәжірибе: өсетін деректерді бөлек коллекцияда сақтау, ал негізгі құжатта тек “ядро” ақпаратты қалдыру. Егер интерфейс соңғы бірнеше жазбаны көрсетуі керек болса, subset pattern қолданылып, соңғы 5–10 элемент preview ретінде сақталады.

Уақыт бойынша топтау

Big Data-да уақытқа байланысты дерек өте көп болады. Мұндай деректер үшін bucket pattern – ең тиімді тәжірибелердің бірі. Идеясы: әр оқиғаны жеке құжатқа шексіз жаза бермей, оларды уақыт интервалымен (сағат/күн/апта) бір “bucket құжатқа” жинақтау немесе керісінше, белгілі бір көлемге жеткенде жаңа bucket ашу. Бұл тәсіл жазуды тұрақтандырады, диск I/O-ны басқаруды жеңілдетеді және “соңғы 7 күн” сияқты сұраныстарды жылдамдатады. Ең бастысы – bucket құжаттарының өлшемі бақыланады, шексіз өсім бір құжатқа жиналмайды.

Индекс стратегиясы: аз, бірақ дәл және пайдалы

Big Data-да индекстерсіз жүйе өмір сүре алмайды, бірақ индекстерді шамадан тыс көбейту де қауіпті. Үздік тәжірибе – индекстерді query patterns-ке сүйеніп таңдау. Әдетте міндетті индекстер: уақыт өрісі (createdAt, timestamp), байланыс өрістері (userId, deviceId, courseId), және жиі фильтрленетін категория/статус өрістері. Ал күрделі тізімдер үшін compound index маңызды: filter + sort сценарийін бір индекспен жабу. Екінші жағынан, әр индекс insert/update кезінде жаңартылатындықтан, write-heavy жүйеде индекстер саны тым көп болса, жазу баяулайды. Сондықтан Big Data үшін “индекс көп” емес, “индекс дәл” деген қағида тиімді.

Дерек типтерін біріздендіру және схема тәртібі

Үлкен жүйеде дерек типтері араласып кетсе, бұл тек “ұқыпсыздық” емес – өнімділік пен сапа проблемасы. Мысалы, бір жерде userId string, бір жерде ObjectId болса, join/lookup бұзылады, фильтр күтпеген нәтиже береді, индекстер тиімді қолданылмауы мүмкін. Сол сияқты уақыт өрістерін бір стильде сақтау керек: бір жерде ISODate, бір жерде string болса, диапазон сұраныстары қиындайды. Үздік тәжірибе – атау стилін (camelCase/snake_case) және тип стандартын бекіту, әрі деректер кіретін жерде валидация қою. Big Data-да кейін “тазарту” өте қымбат, сондықтан тәртіпті бастапқыда енгізген дұрыс.

Жиі кездесетін Big Data қателері және олардан қорғану

Big Data жобаларындағы ең жиі қателердің қатарына мыналар кіреді: өсетін тарихты embedded массивке жинау, many-to-many байланысты үлкен массивтер арқылы сақтау, индекстерсіз filter/sort жасау, типтердің араласуы, денормализацияны бақылаусыз қолдану, $lookup-ты негізгі интерфейсте шамадан тыс пайдалану, және “бір құжатқа тым көп жазу” (hot document). Үздік тәжірибе – осы қауіптерді схема жобалау кезінде-ақ ескеріп, subset/bucket/enrollment сияқты шаблондармен алдын ала шешу.

Практикалық мысал

Егер электронды журнал жүйесі бірнеше жыл жұмыс істесе, бағалар мен қатысу деректері миллиондаған жазбаға жетуі мүмкін. Үздік тәжірибеде студент профилі шағын embedded түрде сақталады, пән/оқытушы reference арқылы бөлек коллекцияда тұрады, ал бағалар/қатысу бөлек коллекцияда уақыт өрісімен жазылады. Соңғы 10 баға превью ретінде subset (ішкі жиын) арқылы профилде көрінуі мүмкін. Уақыт бойынша есептер үшін агрегация қолданылады, ал өте жиі есептер алдын ала есептелген статистика коллекциясына жинақталады. Мұндай архитектура өсуді көтереді және кейін масштабтауға дайын болады.

Қорытынды

Big Data үшін MongoDB схемасын жобалаудың үздік тәжірибелері мына қағидаларға сүйенеді: query-first тәсілі, шексіз өсетін деректерді бөлек сақтау, bucket және subset сияқты шаблондарды қолдану, денормализацияны мақсатпен пайдалану, индекстерді дәл жоспарлау, типтерді біріздендіру және масштабтауға дайын модель құру. Ең бастысы – Big Data жүйесінде “кейін түзетеміз” өте қымбат, сондықтан дұрыс шешімдер бастапқыда қабылдануы тиіс.