Architektura i powłoka

Dokumenty, kolekcje, replica set oraz mongosh.

- MongoDB: dokumenty zamiast tabel
- Architektura MongoDB: dokument, kolekcja, baza, replica set
- mongosh: powłoka administracyjna

MongoDB: dokumenty zamiast tabel

Poziom: podstawy **Dotyczy: MongoDB 7** **Czas: ~10 min**

Ta strona otwiera książkę „MongoDB": skąd wziął się pomysł zapisywania danych jako samodzielnych dokumentów zamiast wierszy w tabelach, jak to wygląda w jednym obrazku i dlaczego w Zielonej stoi na nim katalog produktów sklepu B2B.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
AGPL	Affero General Public License: licencja wolnego oprogramowania, na której MongoDB było wydawane przed przejściem na SSPL w 2018 roku
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
BSON	Binary JSON: binarny format zapisu dokumentów używany wewnętrznie przez MongoDB, w tym w archiwach mongodump
FAQ	Frequently Asked Questions: zbiór najczęściej zadawanych pytań z odpowiedziami
SSPL	Server Side Public License: własna licencja MongoDB Inc. od 2018 roku, utrudniająca dużym dostawcom chmurowym oferowanie MongoDB jako usługi bez osobnej umowy

Skąd się wzięło

> **Pojęcie: baza dokumentowa.** Rodzina silników, w której podstawową jednostką danych jest dokument: samodzielna, ustrukturyzowana porcja danych, mogąca zagnieżdżać w sobie kolejne obiekty i listy, bez rozbijania ich z góry na osobne tabele powiązane kluczami.

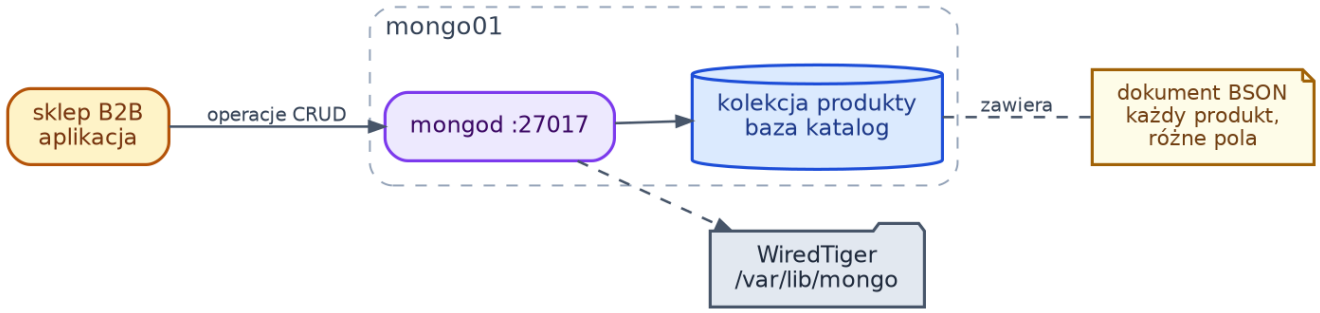
MongoDB powstało w 2007 roku w firmie 10gen, założonej przez byłych pracowników DoubleClick, pierwotnie jako element większej platformy chmurowej podobnej do dzisiejszych usług typu platform as a service. Kiedy sama platforma nie znalazła wystarczającego zainteresowania, zespół 10gen zauważył, że to właśnie baza danych, którą zbudowali na jej potrzeby, jest tym elementem, którym najbardziej interesują się deweloperzy z zewnątrz. Pierwsza publiczna wersja MongoDB

trafiła do użytkowników w lutym 2009 roku, już jako samodzielny produkt open source. Nazwa pochodzi od angielskiego słowa humongous, czyli ogromny, i miała podkreślać, że silnik projektowany jest z myślą o bardzo dużych wolumenach danych od samego początku, nie jako dodatek dorzucony później.

W 2013 roku spółka 10gen zmieniła nazwę na MongoDB Inc., dopasowując tożsamość firmy do produktu, który stał się jej głównym źródłem przychodu. Przez pierwszą dekadę MongoDB rozwijało się na licencji AGPL, ale 8 listopada 2018 roku, wraz z wydaniem 4.0.4, firma przeszła na własną licencję Server Side Public License. Powodem była chęć uniemożliwienia dużym dostawcom chmurowym oferowania MongoDB jako usługi bez wykupienia osobnej umowy z MongoDB Inc. Zmiana okazała się kontrowersyjna na tyle, że MongoDB w tej wersji licencyjnej zniknęło z oficjalnych repozytoriów Debiana, Fedory i Red Hat Enterprise Linux, które uznały SSPL za niespełniającą definicji wolnego oprogramowania.

Jak to działa w jednym obrazku

Aplikacja wysyła operację do procesu `mongod`, który zapisuje albo odczytuje dokument w odpowiedniej kolekcji, bez sprawdzania z góry ustalonego schematu tabeli. Fizycznie dokumenty leżą na dysku w binarnym formacie BSON, zarządzane przez silnik przechowywania WiredTiger.



Aplikacja sklepu B2B zapisuje dokument produktu przez mongod do kolekcji na mongo01 (kliknięcie otwiera pełny rozmiar)

Dlaczego Zielona

> **Z ŻYCIA WZIĘTE**

> Ola przejęła katalog produktów sklepu B2B po firmie, która sprzedała Zielonej gotowe rozwiązanie razem z bazą MongoDB pod spodem, i szybko doceniła, że każdy dostawca w tym katalogu ma inny komplet atrybutów produktu. Jeden przysyła alergeny i termin przydatności, inny wagę i numer partii, a trzeci żadnego z tych pól w ogóle. W bazie relacyjnej każda taka różnica kończyłaby się albo pustymi kolumnami, albo kolejną migracją schematu tabeli.

Katalog produktów na `mongo01`, w bazie `katalog`, wykorzystuje dokładnie tę cechę MongoDB, dla której silnik powstał: dokument każdego produktu może mieć inny zestaw pól, bo różni dostawcy Zielonej po prostu przysyłają różne dane, a aplikacja sklepu B2B czyta z dokumentu to, co akurat w nim jest. Dodanie nowego pola, powiedzmy informacji o certyfikacie ekologicznym dla jednego dostawcy, nie wymaga żadnej migracji całej kolekcji, tylko zaczęcia zapisywania go w nowych dokumentach. Ta swoboda ma swoją cenę: to aplikacja Oli, nie silnik bazy, musi wiedzieć, co zrobić z produktem, który akurat tego pola nie ma.

Mapa tej książki

Książka „MongoDB" składa się z sześciu rozdziałów i zakłada znajomość słownika z książki „Bazy danych: fundamenty".

Rozdział	Na jakie pytanie odpowiada
MongoDB: dokumenty zamiast tabel	Skąd wzięło się MongoDB i dlaczego zmieniło licencję na SSPL
Architektura MongoDB	Co to jest dokument, kolekcja, baza i replica set
mongosh	Jak połączyć się z bazą z linii poleceń
Operacje na dokumentach	Jak wstawiać, zmieniać i usuwać dokumenty
Aggregation pipeline	Jak budować złożone zapytania analityczne
Operatory wyrażeń	Jakich operatorów używamy w zapytaniach i agregacji
Indeksy w MongoDB	Jak przyspieszyć odczyt konkretnych pól
Backup i odtwarzanie	Jak zrobić i odtworzyć kopię zapasową narzędziami mongodump i mongorestore
Upgrade silnika bazy danych	Jak bezpiecznie podnieść wersję MongoDB

Powiązane: Architektura MongoDB · Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin · Upgrade silnika bazy danych

Źródła: Wikipedia, „MongoDB" i „MongoDB Inc." · Wikipedia, „Server Side Public License" · MongoDB, „Server Side Public License FAQ" (mongodb.com) · Percona, „Is MongoDB Truly Open Source? A Critical Look at SSPL"

Architektura MongoDB: dokument, kolekcja, baza, replica set

Poziom: podstawy **Dotyczy: MongoDB 5.0–8.0** **Czas: ~15 min**

Cztery pojęcia, na których stoi MongoDB — dokument, kolekcja, replica set i silnik przechowywania — oraz mapa tego, jak nazywają się te same rzeczy w bazie relacyjnej.

Skróty użyte na tej stronie (5)

Skrót	Co znaczy
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych
BSON	Binary JSON: binarny format zapisu dokumentów używany wewnętrznie przez MongoDB, w tym w archiwach mongodump
JSON	JavaScript Object Notation: tekstowy format danych z klamrami i cudzysłowami
RPO	Recovery Point Objective: ile danych wolno stracić przy awarii
RTO	Recovery Time Objective: jak długo wolno nie działać przy awarii

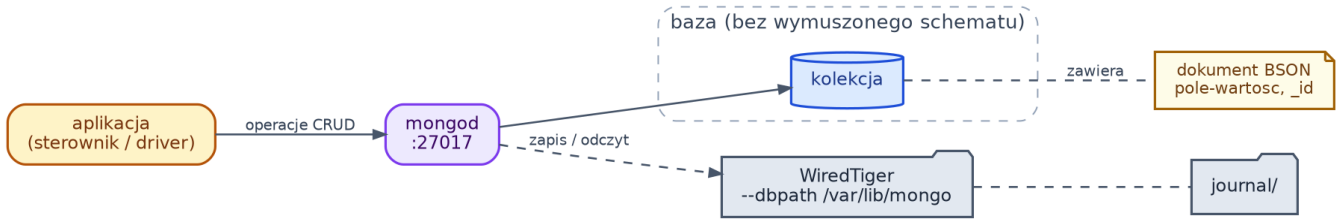
Z życia wzięte

Z ŻYCIA WZIĘTE

O czternastej w piątek Kasia wgrywa nową wersję sklepu B2B i dopisuje pole promocja do części dokumentów w kolekcji produkty na mongo01, nie przygotowując migracji danych historycznych, bo liczy na brak wymuszonego schematu. Strona listingu produktów zaczyna się wywalać dla starszych pozycji katalogu, bo kod frontendu zakłada, że pole promocja zawsze istnieje. Poprawka trafia do warstwy odczytu jako sprawdzenie istnienia pola przed jego użyciem, a nie jako migracja całej kolekcji naraz.

Po co to istnieje

Pojęcie: baza dokumentowa. MongoDB należy do rodziny baz dokumentowych, opisanej ogólnie na stronie Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin — zamiast wierszy w sztywno zdefiniowanych tabelach, podstawową jednostką danych jest tu dokument: samodzielna, ustrukturyzowana porcja danych zapisana w formacie przypominającym JSON, mogąca zagnieżdżać w sobie kolejne obiekty i tablice.



Proces mongod przyjmuje operacje na kolekcji zlozonej z dokumentow BSON i zapisuje je przez WiredTiger na dysk (kliknięcie otwiera pełny rozmiar)

Pojęcie: BSON. Binary JSON — binarny format zapisu, w którym MongoDB faktycznie przechowuje dokumenty na dysku i przesyła je po sieci. Rozszerza składnię JSON o typy, których zwykły JSON nie ma, takie jak data, liczba 64-bitowa czy identyfikator binarny, i jest szybszy do parsowania niż tekstowy JSON, bo długości pól są zakodowane wprost w strumieniu bajtów zamiast wymagać skanowania znak po znaku.

Pojęcie: dokument. Pojedynczy rekord danych, odpowiednik wiersza w bazie relacyjnej, zapisany jako zestaw par pole-wartość. Każdy dokument ma pole `_id`, pełniące rolę klucza głównego, unikalne w obrębie kolekcji — jeśli go nie podamy przy wstawianiu, MongoDB wygeneruje je samo jako `ObjectId`.

Pojęcie: kolekcja. Zbiór dokumentów, odpowiednik tabeli w bazie relacyjnej. W odróżnieniu od tabeli kolekcja nie wymaga z góry zdefiniowanego schematu — nowy dokument może mieć pola, których nie ma żaden inny dokument w tej samej kolekcji.

Brak schematu — co to naprawdę znaczy

> **Pojęcie: schemat.** Ustalona z góry struktura danych: jakie pola ma mieć rekord i jakiego są typu. W silnikach relacyjnych schemat tabeli jest wymuszany przez silnik przy każdym zapisie, MongoDB go nie wymusza.

Brak wymuszonego schematu nie oznacza, że dane w kolekcji faktycznie są przypadkowe. W praktyce niemal każda kolekcja ma swój ustalony kształt, narzucany przez logikę aplikacji, która do niej zapisuje — po prostu ten kształt nie jest zapisany i pilnowany przez silnik bazy, tylko przez kod aplikacji albo, opcjonalnie, przez walidator schematu (`$jsonSchema`) skonfigurowany na poziomie kolekcji.

Konsekwencja praktyczna jest dwojaka. Z jednej strony dodanie nowego pola do dokumentów nie wymaga migracji całej kolekcji, tak jak `ALTER TABLE` w bazie relacyjnej opisany na stronie ALTER TABLE: co blokuje tabelę, a co nie — wystarczy zacząć je zapisywać w nowych dokumentach, a stare po prostu go nie mają. Z drugiej strony ta swoboda przenosi na aplikację odpowiedzialność za spójność, którą w bazie relacyjnej egzekwuje silnik przez `NOT NULL`, `CHECK` czy typy kolumn — kod odczytujący dokumenty musi sam radzić sobie z tym, że pole może nie istnieć, mieć nieoczekiwany typ, albo istnieć tylko w części rekordów historycznych.

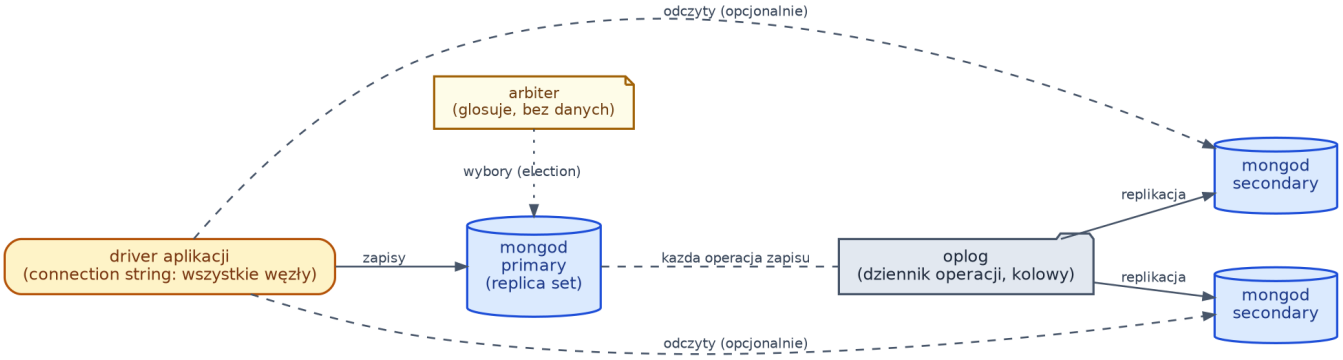
Procesy silnika

> **Pojęcie: mongod.** Główny proces serwera bazy danych — to on przyjmuje połączenia, wykonuje operacje na dokumentach i zarządza danymi na dysku. Pojedyncza instancja `mongod` nasłuchuje domyślnie na porcie `27017` i trzyma swoje pliki danych w katalogu wskazanym parametrem `--dbpath`.

> **Pojęcie: mongos.** Proces routera zapytań, obecny wyłącznie w konfiguracji z shardingiem — sam nie przechowuje żadnych danych, tylko przyjmuje zapytania od klientów i kieruje je do właściwego shardu (fragmentu danych) w klastrze. Aplikacja łączy się zawsze z `mongos`, nigdy bezpośrednio z pojedynczym `mongod` wchodzącym w skład shardowanego klastra.

Replica set

> **Pojęcie: replica set.** Grupa procesów `mongod` trzymających tę samą kopię danych, zapewniająca odporność na awarię pojedynczego serwera. W obrębie replica setu jeden węzeł pełni rolę `primary` i to on jako jedyny przyjmuje operacje zapisu; pozostałe węzły, `secondary`, replikują zapisy z primary asynchronicznie i mogą (jeśli aplikacja tak skonfiguruje odczyt) obsługiwać zapytania odczytu.



Primary przyjmuje zapisy i zapisuje je w oplogu, sekundary replikuja z niego, driver laczy sie z calym replica setem (kliknięcie otwiera pełny rozmiar)

Gdy primary przestaje odpowiadać, pozostałe węzły automatycznie przeprowadzają wybory (election) i jeden z secondary zostaje nowym primary — mechanizm ten wymaga nieparzystej liczby głosujących węzłów, żeby uniknąć remisu. Jeśli liczba pełnoprawnych węzłów danych wychodzi parzysta, dokłada się węzeł `arbiter` — proces uczestniczący w głosowaniu, ale nie trzymający kopii danych — wyłącznie po to, żeby zapewnić nieparzystość głosów.

Sharding w skrócie

> **Pojęcie: sharding.** Mechanizm poziomego podziału danych jednej kolekcji na kilka niezależnych serwerów (shardów), stosowany, gdy dane albo obciążenie zapytaniami przestają mieścić się na pojedynczym replica secie. Każdy shard jest sam w sobie osobnym replica setem, a to, który dokument trafia na który shard, decyduje wartość wybranego pola zwanego kluczem shardowania. Klaster shardowany dokłada dwa dodatkowe elementy nieobecne w pojedynczym replica secie: proces routera `mongos`, przez który przechodzą

wszystkie zapytania, oraz `config servers`, osobny replica set trzymający metadane o tym, który zakres kluczy shardowania leży na którym shardzie.

Analogia

Przejście od bazy relacyjnej do dokumentowej najłatwiej zobaczyć na przykładzie segregatora i teczki.

Baza relacyjna to segregator z gotowymi, jednakowymi formularzami — każda kartka (wiersz) ma z góry wydrukowane te same rubryki (kolumny), niezależnie od tego, czy dana sprawa akurat te rubryki wypełnia. Chcąc dopisać nową rubrykę do formularza, trzeba przedrukować cały nakład — to jest `ALTER TABLE` na dużej tabeli, operacja kosztowna właśnie dlatego, że dotyczy wszystkich kartek naraz. Baza dokumentowa to raczej teczka z osobnymi kartkami swobodnego formatu — jedna sprawa ma dołączony skan, inna ma dopisaną notatkę odręczną, a jeszcze inna nie ma nic poza samym wnioskiem. Nikt nie przedrukowuje całej teczki, żeby dodać nowy rodzaj załącznika do jednej sprawy.

Granica tej wygody jest taka sama jak granica samej teczki: łatwo w niej znaleźć kartkę po numerze sprawy przypiętym z góry (odpowiednik `_id`), trudniej szybko odpowiedzieć na pytanie w stylu „ile spraw ma załącznik typu X", bo nikt nie prowadził osobnego rejestru tego pola — dokładnie to zadanie przejmuje w MongoDB indeks założony świadomie na tym polu, opisany na stronie Indeksy w MongoDB.

Silnik przechowywania

Pojęcie: WiredTiger. Domyślny silnik przechowywania danych MongoDB od wersji 3.2 — odpowiada za to, jak dokumenty są faktycznie zapisywane i odczytywane z dysku, jak działa kompresja i jak przebiegają blokady na poziomie dokumentu. WiredTiger trzyma dane w katalogu wskazanym `--dbpath`, domyślnie `/var/lib/mongo` na dystrybucjach linuksowych, w plikach o rozszerzeniu `.wt` — po jednym na kolekcję i po jednym na każdy jej indeks, plus dziennik zapisu (`journal`) w podkatalogu `journal/`, zapewniający odtworzenie niezatwierdzonych jeszcze na dysk zmian po awarii.

Jak to działa: droga zapytania w replica secie

Klient aplikacyjny łączy się z całym replica setem naraz, podając w connection stringu adresy wszystkich jego węzłów, nie jednego wybranego. Sterownik (driver) MongoDB po stronie aplikacji sam ustala, który węzeł jest aktualnie `primary`, i kieruje do niego wszystkie operacje zapisu — próba zapisu wysłana wprost do węzła `secondary` kończy się odrzuceniem, bo `secondary` z definicji tylko odczytuje zmiany replikowane z `primary`, nigdy nie przyjmuje ich bezpośrednio od klienta.

Operacja zapisu na `primary` trafia najpierw do lokalnego zbioru danych węzła, a zaraz potem zostaje odnotowana w jego `oplogu`, opisanym na stronie Backup i odtwarzanie: mongodump i mongorestore — wewnętrznym, kołowym dzienniku operacji. Każdy `secondary` stale odczytuje ten dziennik z `primary` i powtarza u siebie te same operacje, w tej samej kolejności. To, jak bardzo świeże dane zwróci zapytanie odczytu skierowane do `secondary`, zależy więc od tego, jak daleko dany węzeł zdążył nadążyć za odczytem oplogu — opóźnienie to nazywa się replication lag i rośnie zwykle przy dużym obciążeniu zapisem albo przy słabszym łączu sieciowym do danego węzła.

Trwałość zapisu: write concern

> **Pojęcie: write concern.** Poziom potwierdzenia, jakiego aplikacja żąda od serwera, zanim operacja zapisu zostanie uznana za zakończoną powodzeniem. Domyślny poziom `w: 1` uznaje zapis za potwierdzony, gdy przyjmie go węzeł `primary`, jeszcze zanim zdąży go zreplikować do jakiegokolwiek `secondary` — najszybszy wariant, ale w razie awarii `primary` tuż po takim zapisie, zanim replikacja nadąży, dana zmiana może zostać utracona. Poziom `w: "majority"` każe czekać na potwierdzenie zapisu przez większość głosujących węzłów replica setu, kosztem nieco większego opóźnienia, za to z gwarancją, że zapis przetrwa utratę pojedynczego węzła, łącznie z samym `primary`.

Wybór między tymi poziomami jest właśnie tym, o czym mówi ogólna reguła RPO i RTO z fundamentów, opisana na stronie RPO i RTO: dwie liczby, od których zaczyna się każdy plan backupu — `w: 1` obniża opóźnienie zapisu, `w: "majority"` obniża ryzyko utraty danych przy awarii.

Mapa pojęć: relacyjne na dokumentowe

Relacyjnie	Dokumentowo (MongoDB)
tabela	kolekcja
wiersz	dokument
kolumna	pole
klucz główny	`_id`
indeks	indeks (`createIndex`, opisany na stronie Indeksy w MongoDB)
`JOIN`	`$lookup` w potoku agregacji, opisanym na stronie Aggregation pipeline
schemat tabeli	brak wymuszonego schematu (opcjonalny `$jsonSchema`)
transakcja wielotabelowa	transakcja wielodokumentowa (od MongoDB 4.0)

Częste błędy

Traktowanie braku schematu jako braku struktury. Kolekcja bez wymuszonego schematu wciąż ma swój faktyczny kształt narzucony przez aplikację — zmiana tego kształtu bez kontrolowanej migracji danych historycznych prowadzi do dokumentów, które trzeba obsługiwać dwoma różnymi ścieżkami kodu na raz.

Zakładanie, że `$lookup` działa jak `JOIN` bez konsekwencji wydajnościowych. `$lookup` w potoku agregacji jest operacją kosztowną, bo dla każdego dokumentu wejściowego przeszukuje osobno kolekcję docelową — model dokumentowy premiuje raczej zagnieżdżanie powiązanych danych wewnątrz jednego dokumentu niż łączenie wielu kolekcji przy każdym odczycie.

Powiązane: Relacyjne, dokumentowe, kolumnowe, klucz-wartość: mapa rodzin · mongosh: powłoka administracyjna · Aggregation pipeline · Indeksy w MongoDB

Źródła: notatki własne administratora MongoDB · dokumentacja MongoDB, rozdział „MongoDB Manual — Introduction" i „Replication"

mongosh: powłoka administracyjna

Poziom: podstawy **Dotyczy: MongoDB 5.0–8.0, mongosh** **Czas: ~20 min**

Interaktywna powłoka administracyjna oparta na JavaScripcie, przez którą wykonujemy niemal każdą operację administracyjną na MongoDB — logowanie, przegląd baz, statystyki i diagnostykę bieżącej aktywności.

Skróty użyte na tej stronie (1)

Skrót	Co znaczy
B2B	business to business: sprzedaż i systemy skierowane do innych firm, nie do klientów indywidualnych

Fiszka

TYLKO ODCZYT

Uruchomienie powłoki bezpośrednio na hoście

```
mongosh
```

Łączymy się z domyślnym serwerem lokalnym na `localhost:27017` bez uwierzytelniania. Po połączeniu pojawia się znak zachęty, w który wpisujemy kolejne komendy.

Przykład:

```
Current Mongosh Log ID: 66f1a2b3c4d5e6f7a8b9c0d1
Connecting to:          mongodb://127.0.0.1:27017/?directConnection=true
Using MongoDB:          7.0.14
Using Mongosh:          2.3.1
```

```
test>
```

mongosh = Mongo shell: klient MongoDB

TYLKO ODCZYT

Uruchomienie powłoki wewnątrz kontenera

```
sudo docker exec -it NazwaKontenera mongosh
```

Uruchamiamy powłokę wewnątrz działającego kontenera zamiast na hoście, gdy serwer MongoDB działa w Dockerze. Zasada przekładu jest stała: dowolną komendę `mongosh ...` z tej strony można uruchomić w kontenerze, poprzedzając ją `docker exec -it NazwaKontenera` w trybie interaktywnym albo samym `docker exec NazwaKontenera` przy jednolinijkowcach z `--eval`.

docker = narzędzie do kontenerów

TYLKO ODCZYT

Połączenie z uwierzytelnieniem

```
mongosh -u NazwaUsera -p --authenticationDatabase admin
```

Podajemy nazwę użytkownika flagą `-u`, a flaga `-p` bez wartości wymusza interaktywny monit o hasło zamiast wpisywania go jawnie w linii poleceń. Flaga `--authenticationDatabase` wskazuje bazę, w której zdefiniowano to konto — konta administracyjne żyją zwykle w bazie `admin`, nawet jeśli mają uprawnienia do zupełnie innych baz danych.

mongosh = Mongo shell: klient MongoDB

TYLKO ODCZYT

Wykonanie pojedynczej komendy bez wchodzenia do powłoki

```
mongosh --quiet --eval 'db.version()'
```

Wykonujemy jeden fragment kodu i kończymy działanie, bez wchodzenia w tryb interaktywny — przydatne w skryptach powłoki i zadaniach cyklicznych. Flaga `--quiet` wycisza baner powitalny, który inaczej trafiłby razem z wynikiem do przetwarzanego dalej tekstu.

Przykład:

```
7.0.14
```

mongosh = Mongo shell: klient MongoDB

TYLKO ODCZYT

Wylistowanie wszystkich baz danych: show dbs

```
show dbs
```

Wyświetlamy listę wszystkich baz danych na serwerze wraz z ich rozmiarem na dysku. Bazy zupełnie puste, bez żadnych zapisanych dotąd danych, mogą się na tej liście nie pojawić.

Przykład:

```
test> show dbs
admin    180.00 KiB
appdb    24.61 MiB
config    60.00 KiB
local     72.00 KiB
```

show = FortiOS: wypisuje konfigurację, w podstawowej postaci tylko ustawienia różniące się od domyślnych, a z wariantem full-configuration także wartości domyślne

TYLKO ODCZYT

Wylistowanie wszystkich baz z poziomu kontenera

```
sudo docker exec -it NazwaKontenera mongosh --eval
"db.getMongo().getDBNames().forEach(function(db) { print(db) });"
```

Wypisujemy same nazwy baz danych, jedna pod drugą, bez wchodzenia do interaktywnej powłoki wewnątrz kontenera — wygodne do wykorzystania dalej w skrypcie powłoki hosta, na przykład jako lista do przeiterowania kolejnymi backupami.

Przykład:

```
admin
appdb
config
local
```

docker = narzędzie do kontenerów

TYLKO ODCZYT

Przełączenie na inną bazę danych: use

```
use appdb
```

Przełączamy kontekst na wskazaną bazę. Od tego momentu zmienna `db` w powłoce odnosi się właśnie do niej. Jeśli baza jeszcze nie istnieje, zostanie faktycznie utworzona dopiero przy pierwszym zapisie do niej jakiegokolwiek dokumentu.

TYLKO ODCZYT

Wylistowanie kolekcji bieżącej bazy: show collections

```
show collections
```

Wyświetlamy wszystkie kolekcje istniejące w bazie, na którą jesteśmy aktualnie przełączeni poleceniem `use`.

Przykład:

```
appdb> show collections
klienci
zamowienia
produkty
```

show = FortiOS: wypisuje konfigurację, w podstawowej postaci tylko ustawienia różniące się od domyślnych, a z wariantem full-configuration także wartości domyślne

TYLKO ODCZYT

Sprawdzenie statystyk bieżącej bazy: db.stats()

```
db.stats()
```

Zwracamy statystyki bieżącej bazy danych — liczbę kolekcji, liczbę dokumentów łącznie i zajętość miejsca na dysku. Pomocne przy ocenie rozmiaru bazy przed wykonaniem kopii zapasowej opisanej na stronie Backup i odtwarzanie: mongodump i mongorestore.

Przykład:

```
{
  db: 'appdb',
  collections: 3,
  views: 0,
  objects: 148213,
  avgObjSize: 512.4,
  dataSize: 75923456,
  storageSize: 40894464,
  indexes: 5,
  indexSize: 3096576,
  totalSize: 43991040,
  ok: 1
}
```

TYLKO ODCZYT

Sprawdzenie statystyk pojedynczej kolekcji: db.collection.stats()

```
db.zamowienia.stats()
```

Zwracamy te same statystyki, ale zawężone do jednej wskazanej kolekcji, wraz z rozmiarem każdego indeksu założonego na niej z osobna.

Przykład:

```
{
  ns: 'appdb.zamowienia',
  size: 51203840,
  count: 98211,
  avgObjSize: 521,
  storageSize: 28901376,
  nindexes: 2,
  indexSizes: { _id_: 1638400, klient_id_1: 921600 },
  ok: 1
}
```

🔒 TYLKO ODCZYT

Sprawdzenie statusu całego serwera: db.serverStatus()

```
db.serverStatus()
```

Zwracamy obszerny dokument z metrykami całego serwera — liczbą aktywnych połączeń, zużyciem pamięci, licznikami operacji i stanem replikacji. Wynik jest zwykle bardzo duży, dlatego w praktyce częściej odczytujemy pojedyncze pola z niego, na przykład `db.serverStatus().connections`.

Przykład:

```
appdb> db.serverStatus().connections
{
  current: 14,
  available: 51186,
  totalCreated: 302,
  active: 3
}
```

🔒 TYLKO ODCZYT

Sprawdzenie aktualnie trwających operacji: db.currentOp()

```
db.currentOp()
```

Zwracamy listę wszystkich operacji aktualnie wykonywanych przez serwer — jej numer (`opid`), czas trwania w sekundach i treść zapytania. Sięgamy po to, gdy podejrzewamy zapytanie, które wisi dłużej, niż powinno.

Przykład:

```
{
  inprog: [
    {
      opid: 4821059,
      secs_running: 42,
      op: 'query',
      ns: 'appdb.zamowienia',
```

```
      command: { find: 'zamowienia', filter: { status: 'oplacone' } }
    }
  ]
}
```

TYLKO ODCZYT

Zakończenie długo trwającej operacji: db.killOp()

```
db.killOp(4821059)
```

Przerywamy operację o wskazanym numerze `opid`, odczytanym wcześniej z `db.currentOp()`. Używamy tego, gdy zapytanie blokuje inne operacje albo trwa wyraźnie dłużej, niż zakładaliśmy.

TYLKO ODCZYT

Sprawdzenie stanu replica setu: rs.status()

```
rs.status()
```

Zwracamy stan całego replica setu — który węzeł jest aktualnie `PRIMARY`, które są `SECONDARY`, jak dawno każdy z nich potwierdził swoją obecność (`heartbeat`) i jak duże jest opóźnienie replikacji między nimi.

Przykład:

```
{
  set: 'rsAppdb',
  members: [
    { _id: 0, name: 'mongo01:27017', stateStr: 'PRIMARY', health: 1 },
    { _id: 1, name: 'mongo02:27017', stateStr: 'SECONDARY', health: 1 }
  ],
  ok: 1
}
```

Z życia wzięte

Z ŻYCIA WZIĘTE

O dziesiątej rano Kasia zgłasza, że strona kategorii w sklepie B2B ładuje się kilkanaście sekund zamiast ułamka sekundy. Tomek loguje się mongoshem na mongo01 i sprawdza bieżące operacje, znajdując zapytanie do kolekcji produkty, które trwa już ponad minutę i skanuje całą kolekcję zamiast korzystać z indeksu. Zabija tę operację, żeby zwolnić serwer, a dopiero potem szuka w kodzie aplikacji, skąd wzięło się zapytanie bez odpowiedniego filtra po polu z indeksem.

Po co to istnieje

Pojęcie: mongosh. Powłoka `mongosh` zastąpiła starszą `mongo`, wycofaną w MongoDB 6.0, i pozostaje głównym narzędziem administracyjnym do pracy z serwerem zarówno z poziomu skryptów, jak i sesji interaktywnej. Sposób wejścia do powłoki zależy od tego, gdzie faktycznie działa serwer — bezpośrednio na hoście czy wewnątrz kontenera — natomiast wszystkie komendy wykonywane już wewnątrz powłoki (zaczynające się od `db.` albo od `show`) wyglądają identycznie w obu przypadkach.

Częste błędy

Podanie hasła wprost w linii poleceń. `mongosh -u NazwaUsera -p HasłoWprost` zostawia hasło w historii powłoki i w wyniku `ps aux` widocznym dla innych kont na tym samym hoście — flaga `-p` bez wartości, wymuszająca monit interaktywny, jest bezpieczniejsza.

Odczyt `db.serverStatus()` w całości zamiast pojedynczego pola. Wynik bywa na tyle obszerny, że przewijanie go w poszukiwaniu jednej metryki jest wolniejsze niż odwołanie się od razu do konkretnego pola dokumentu.

Powiązane: Architektura MongoDB: dokument, kolekcja, baza, replica set · Operacje na dokumentach · Backup i odtwarzanie: mongodump i mongorestore

Źródła: notatki własne administratora MongoDB · dokumentacja MongoDB, rozdział „mongosh" i „Database Commands"