UNIVERZITA PARDUBICE   
Fakulta elektrotechniky a informatiky

*MongoDB Cluster*

*Andrii Streblychenko 3 ročník*

V …..dne….

Obsah

[Úvod 4](#_Toc195012567)

[1 Architektura 5](#_Toc195012568)

[1.1 Schéma a popis architektury 5](#_Toc195012569)

[1.2 Specifika konfigurace 6](#_Toc195012570)

[1.2.1 CAP teorém 6](#_Toc195012571)

[1.2.2 Cluster 6](#_Toc195012572)

[1.2.3 Uzly 6](#_Toc195012573)

[1.2.4 Sharding 6](#_Toc195012574)

[1.2.5 Replikace 7](#_Toc195012575)

[1.2.6 Perzistence dat 7](#_Toc195012576)

[1.2.7 Distribuce dat 7](#_Toc195012577)

[1.2.8 Zabezpečení 7](#_Toc195012578)

[2 Funkční řešení 8](#_Toc195012579)

[2.1 Struktura 8](#_Toc195012580)

[2.1.1 docker-compose.yml 9](#_Toc195012581)

[2.2 Instalace 10](#_Toc195012582)

[3 Případy užití a případové studie 11](#_Toc195012583)

[3.1 Účely použití databáze MongoDB 11](#_Toc195012584)

[3.2 Důvody výběru databáze MongoDB pro tento projekt 11](#_Toc195012585)

[3.3 Případové studie 11](#_Toc195012586)

[3.3.1 Vývoj e-commerce platformy pomocí MERN stacku 11](#_Toc195012587)

[3.3.2 Využití MongoDB v IBM pro správu různorodých dat 12](#_Toc195012588)

[3.3.3 Definice a vlastnosti MongoDB podle TechTarget 12](#_Toc195012589)

[4 Výhody a nevýhody 13](#_Toc195012590)

[4.1 Výhody a nevýhody MongoDB 13](#_Toc195012591)

[4.2 Výhody a nevýhody navrženého řešení 13](#_Toc195012592)

[5 Další specifika 15](#_Toc195012593)

[6 Data 16](#_Toc195012594)

[6.1 Použité datové soubory 16](#_Toc195012595)

[6.2 Struktura a typy dat 16](#_Toc195012596)

[6.3 Rozsah dat 16](#_Toc195012597)

[6.4 Úpravy dat 17](#_Toc195012598)

[6.5 Zdroj dat 17](#_Toc195012599)

[6.6 Analýza v Pythonu 17](#_Toc195012600)

[7 Dotazy 18](#_Toc195012601)

[Závěr 19](#_Toc195012602)

[Zdroje 20](#_Toc195012603)

[Přílohy 21](#_Toc195012604)

Úvod

xxxxxxxxx

# Architektura

Cílem této kapitoly je detailně popsat architekturu navrženého řešení pomocí dokumentové NoSQL databáze **MongoDB** ve formě plně shardovaného a replikovaného clusteru. Tento systém je nasazen pomocí **Docker Compose** a využívá více kontejnerů pro různé role v clusteru, jako jsou **config servery, datové shardy, mongos routovací uzly** a **NGINX load balancer**. Řešení je navrženo s důrazem na škálovatelnost, vysokou dostupnost a bezpečnost komunikace mezi jednotlivými komponentami. Architektura reflektuje doporučené praktiky MongoDB

## Schéma a popis architektury

Изображение выглядит как текст, снимок экрана, дисплей, диаграмма

Контент, сгенерированный ИИ, может содержать ошибки.

Celý systém je navržen jako plně škálovatelný MongoDB sharded cluster, nasazený prostřednictvím docker-compose. Architektura zahrnuje tyto komponenty:

NGINX Load Balancer (27080). NGINX slouží jako vstupní bod zvenku. Rozkládá zátěž mezi dvě instance mongos routeru, čímž zajišťuje vysokou dostupnost a vyrovnávání zátěže.

Routers – mongos1, mongos2. Mongos je komponenta MongoDB, která směruje dotazy na správné shardy podle metadata uložených v config servers. Dvě instance mongos zajišťují redundanci a lepší výkon.

Config Servers – configsvr1, configsvr2, configsvr3. Tyto tři uzly tvoří replikový set configReplSet, který uchovává metadata o rozmístění dat mezi shardami. Tato komponenta je nezbytná pro správnou funkčnost sharded clusteru.

Shards – shard1, shard2, shard3. Každý shard je samostatný replikový set o třech uzlech – jeden hlavní (PRIMARY) a dva sekundární (SECONDARY). Tato architektura zajišťuje vysokou dostupnost a replikaci dat pro každý shard.

Docker síť. Všechny komponenty běží v izolované Docker síti, což umožňuje vzájemnou komunikaci, ale omezuje přístup zvenku pouze na port 27080 skrze NGINX.

Odlišnosti od doporučené architektury MongoDB:

Byla přidána NGINX vrstva jako externí přístupový bod s vyrovnáváním zátěže.

## Specifika konfigurace

Architektura tohoto MongoDB řešení je navržena jako plně shardovaný a replikovaný cluster s důrazem na škálovatelnost, vysokou dostupnost, bezpečnost a rozšiřitelnost. Všechny komponenty jsou nasazeny prostřednictvím Docker Compose a komunikují spolu ve vnitřní síti Dockeru.

### CAP teorém

MongoDB ve sharded architektuře se přiklání ke kombinaci CP (Consistency + Partition Tolerance)

Consistency: MongoDB replikové sety zaručují, že zápisy jsou konzistentní a dostupné na PRIMARY uzlu.

Partition Tolerance: systém je odolný vůči výpadkům částí sítě nebo uzlů díky shardingu a replikaci.

### Cluster

Používá se jeden cluster MongoDB, postaveny na «*docker-compose*», který obsahuje všechny komponenty (config servery, mongos, shardy, load balancer). Tento přístup odpovídá standardnímu návrhu distribuované MongoDB infrastruktury.

### Uzly

Celkem je nasazeno 14 uzlů MongoDB:

**3 config servery + 2 mongos routery + (3 shardy × 3 repliky) = 14 uzlů**

Každý replikový set má jednoho PRIMARY a dva SECONDARY uzly.

### Sharding

Použity jsou 3 shardy: *shard1, shard2, shard3* Každý shard obsahuje 3 repliky. Výběr byl učiněn kvůli práci s rozsáhlými daty (např. tisíce záznamů o přestupcích a transakcích), kde je důležité rovnoměrné rozložení zátěže a výkon.

### Replikace

Každý shard je implementován jako replikový set se třemi uzly – *PRIMARY* a dvěma *SECONDARY*. Tim se dava ochranu před výpadkem jednoho nebo dvou uzlů, možnost čtení z replik, pokud je potřeba a automatické přepnutí PRIMARY uzlu v případě výpadku.

### Perzistence dat

Data se ukládají do Docker volume (mapované na /data/dbX) na každém uzlu. MongoDB používá WiredTiger storage engine, který zajišťuje zápis na disk při každé operaci (write concern) a journaling pro zotavení při selhání. Data jsou perzistentní i při restartu kontejnerů díky svázaným svazkům (volumes).

### Distribuce dat

MongoDB rozděluje data na základě shard key, které určujeme při vytváření kolekcí. *Config servery* drží metadata o rozmístění chunků (částí kolekcí). *Mongos* router rozhoduje, který shard má být dotazován. Díky replikaci jsou zápisy zajištěny i v případě výpadku jednotlivého uzlu.

### Zabezpečení

Používá se keyfile-based internal authentication – všechny uzly (config servery, mongos, shard) sdílejí jeden *keyfile*. Dále je nastaven uživatel *admin* s heslem přes .*env* soubor, vyžadovaný pro všechny externí přístupy. Všechny porty kromě *27080* (NGINX) jsou uzavřeny pro vnější přístup, což chrání před neautorizovanými přístupy.

# Funkční řešení

Cílem projektu bylo vytvořit plně automatizovaný a škálovatelný MongoDB cluster, který by fungoval „z krabice“ – tedy bez nutnosti manuálního nastavování jednotlivých komponent. Díky tomu celý cluster lze spustit jediným příkazem přes Docker Compose. Toho bylo dosaženo kombinací konfigurací v Docker Compose souboru (docker-compose.yml), Dockerfile-ů a Bash skriptů (entrypoint skripty, inicializační skripty shardů a config serverů, a automatická tvorba validačních schémat kolekcí). V případě potřeby rozšíření clusteru stačí přidat nové uzly do existujících skriptů a Docker konfigurací.

## Struktura

* **mongodb\_cluster**
  + **config-server**
    - Dockerfile – definuje obraz Docker kontejneru config serveru.
    - configsvr.conf – konfigurační soubor MongoDB config serveru.
    - entrypoint.sh – skript, který inicializuje config server.
    - keyfile – sdílený soubor pro interní autentizaci clusteru.
  + **mongos**
    - Dockerfile – definuje obraz Docker kontejneru pro mongos router.
    - mongos.conf – konfigurační soubor pro mongos router.
    - entrypoint.sh – automaticky registruje shardy v clusteru.
    - schema-collections-init.sh – automatická inicializace validačních schémat kolekcí.
    - keyfile – sdílený autentizační soubor.
  + **shard**
    - Dockerfile – definuje obraz Docker kontejneru shardu s replikační sadou.
    - entrypoint.sh – inicializační skript shardu, který startuje všechny instance shardu.
    - init-shard.sh – nastavuje MongoDB replikační sadu pro shard.
    - mongod{1-3}.conf.template – šablony konfigurací pro jednotlivé instance shardu.
    - supervisord.conf.template – konfigurace Supervisora pro správu instancí shardu.
    - keyfile – sdílený autentizační soubor.
  + **nginx**
    - nginx.conf – konfigurace load balanceru, který směruje požadavky na mongos routery.
  + .env – definuje přístupové údaje a společné proměnné prostředí.
  + docker-compose.yml – centrální soubor konfigurující celý cluster MongoDB.

### docker-compose.yml

Soubor docker-compose.yml definuje strukturu a konfiguraci celého clusteru MongoDB v Docker kontejnerech. Cluster je složen z:

**Config servery** (3 uzly: configsvr1, configsvr2, configsvr3)

* Uložení metadat a konfigurace celého sharded clusteru.
* Nastavují se v režimu replikační sady nazvané configReplSet.
* Používají se sdílené proměnné prostředí (.env) a keyfile pro zabezpečenou interní komunikaci.
* Uzly sdílí volume pro perzistenci konfiguračních dat.

**Shardy** (3 replikované shardy: shard1, shard2, shard3)

* Každý shard sestává ze 3 instancí (hlavní uzel + 2 repliky).
* Každý shard má vlastní replikační sadu (např. shard1).
* Každá instance shardu běží pod kontrolou Supervisoru.
* Data jsou persistována přes oddělené Docker volumes.

**Mongos routery** (2 routery: mongos1, mongos2)

* Přijímají a směrují dotazy aplikací na konkrétní shardy.
* První router automaticky registruje shardy při spuštění (REGISTER\_SHARDS=true).
* Komunikují s config servery a shardy přes interní Docker síť.

**NGINX load balancer**

* Veřejně přístupný jediný bod vstupu (port 27080).
* Směruje požadavky na oba mongos routery pro zajištění vysoké dostupnosti a škálovatelnosti.
* Konfigurační soubor NGINX se připojuje přes volume.

Díky této konfiguraci všechny služby běží izolovaně a bezpečně. Jediný veřejně dostupný port je NGINX na localhost:27080.

## Instalace

Instalace řešení byla vytvořena tak, aby probíhala zcela automaticky, s minimální nutností manuálních zásahů. Celý cluster MongoDB lze spustit jediným příkazem z kořenového adresáře projektu:

**docker-compose up --build**

Parametr --build zajistí automatické sestavení všech kontejnerů a spuštění jejich inicializačních skriptů, čímž nastaví replikační sady, autentizaci a registraci shardů v clusteru.

Cluster bude po úspěšném spuštění dostupný přes load balancer na:

**mongodb://clusterAdmin:YourStrongPassword@localhost:27080**

* Uživatel: clusterAdmin
* Heslo: YourStrongPassword
* Připojení: Port 27080, databáze: main

Databáze main se zakládá automaticky při spuštění clusteru, spolu s kolekcemi a jejich validačními schématy.

Testovací data je možné nahrát následně jednoduchým spuštěním připravených Python skriptů, které jsou umístěné v adresáři noSQL/Data/scripts. Tyto skripty automaticky nahrají CSV datasety do odpovídajících MongoDB kolekcí:

**python upload\_netflix\_films.py**

**python upload\_indian\_traffic.py**

**python upload\_mastercard\_stock.py**

# Případy užití a případové studie

## Účely použití databáze MongoDB

MongoDB je optimální volbou pro aplikace vyžadující vysokou flexibilitu při práci s daty, rychlé dotazování a snadnou rozšiřitelnost. Tyto schopnosti plynou z její dokumentově orientované architektury, která používá formát BSON – binární rozšíření JSON. Díky této struktuře není nutné předem definovat pevné schéma a data lze upravovat i během provozu.

Systémy pracující s obsahem, profitují ze schopnosti MongoDB efektivně ukládat nestrukturované i polostrukturované informace. V oblasti e-commerce umožňuje databáze rychlé operace se sklady, produkty i zákaznickými daty. IoT řešení a mobilní aplikace získávají možnost dynamického ukládání senzorických dat bez nutnosti konverze do tabulkového formátu. Webové platformy ukládají profily, komentáře či hodnocení s minimální latencí. MongoDB také nachází uplatnění při analýze dat v reálném čase – například v monitorovacích nástrojích a vizualizačních panelech typu dashboard.

## Důvody výběru databáze MongoDB pro tento projekt

MongoDB byla zvolena jako hlavní databázová platforma kvůli potřebě pracovat s rozsáhlými, různorodými daty bez nutnosti pevně definované struktury. Projekt vyžaduje schopnost snadno importovat a analyzovat tisíce záznamů z různých domén – například dopravní přestupky, historické burzovní hodnoty nebo filmové záznamy.

Volba padla na MongoDB i kvůli osobní preferenci pracovat s moderní technologií, která je široce využívána v praxi a poptávaná na trhu práce. Apache Cassandra je orientovaná na sloupcová data a její model je zaměřený především na velmi rychlé zápisy a čtení v masivně distribuovaném prostředí a v tomto případě přinášelo omezení kvůli nedostatku flexibility při práci s dynamickými schématy. Databáze typu klíč-hodnota (Redis, Valkey) jsou vhodné spíše pro krátkodobá data nebo cache, nikoli pro komplexní datové modely. Taky s Redis jsem už pracoval.

Řešení bylo navrženo tak, aby podporovalo horizontální škálování, automatizaci nasazení a rychlou odezvu systému. MongoDB umožňuje efektivní využití shardingu, replikace i indexování. V praxi by tento systém mohl být nasazen například v analytických firmách zaměřených na business intelligence, kde je prioritou rychlá transformace surových dat do využitelných informací.

## Případové studie

### Vývoj e-commerce platformy pomocí MERN stacku

Aplikace pro online prodej dokáže zajistit rychlou obsluhu uživatelů díky efektivní správě dat o zákaznících, objednávkách a produktech. Tuto schopnost zajišťuje použití MongoDB v rámci technologie MERN (MongoDB, Express, React, Node). Studie autorů Priyama Joshiho a Bikrama Sarkara z Parul University popisuje systém, který zpracovává objednávky, odesílá automatizované e-maily a nabízí transakční přehledy v reálném čase. MongoDB zde pomáhá s udržením výkonu díky schopnosti flexibilního ukládání různorodých dokumentů a snadné škálovatelnosti [1].

### Využití MongoDB v IBM pro správu různorodých dat

IBM využívá MongoDB při správě rozsáhlých a různorodých datových sad. MongoDB zde slouží jako nástroj, který nahrazuje tradiční tabulkové systémy dokumenty, čímž zjednodušuje strukturu databází a zrychluje vývoj. Databáze umožňuje IBM vyvíjet aplikace pro různé platformy bez nutnosti složitých konverzí dat a zároveň poskytuje prostředí pro škálovatelnou a flexibilní práci s daty [2].

### Definice a vlastnosti MongoDB podle TechTarget

Organizace, které potřebují pracovat s velkým objemem distribuovaných dat, nacházejí v MongoDB vhodné řešení díky jejímu výkonu a možnostem přizpůsobení. TechTarget popisuje MongoDB jako nástroj, který podporuje ad-hoc dotazy, automatické vyvažování zátěže, agregace a výkonné indexování. To vše z ní činí preferovanou databázi v projektech s vysokými nároky na variabilitu a škálovatelnost dat [3].

# Výhody a nevýhody

## Výhody a nevýhody MongoDB

***Výhody***

1. Práce s různorodými a měnícími se datovými strukturami je v MongoDB bezproblémová díky volnému schématu a formátu BSON.
2. Vysoká rychlost vývoje je umožněna flexibilitou dotazovacího jazyka, který nabízí agregace, vnořené výrazy a dynamické filtry.
3. Rozšiřování úložiště je snadno dosažitelné pomocí shardingu, který rozděluje data horizontálně mezi více serverů.
4. Široká škála případů použití je umožněna podporou více datových modelů (dokumentový, key-value, grafový) v jednom systému.

***Nevýhody***

1. Zpracování komplexních relací mezi daty je neefektivní kvůli absenci plnohodnotných JOIN operací.
2. Nepřesně navržená validace může vést k nesourodým strukturám, protože MongoDB nevyžaduje explicitní schéma.
3. Při masivních zápisech dochází ke kolísání výkonu, pokud nejsou optimálně nakonfigurovány indexy a distribuce dat.
4. Obnova a zálohování dat z více shardovaných a replikovaných nodů je složitější než v monolitických databázových systémech.

## Výhody a nevýhody navrženého řešení

***Výhody***

1. Nasazení celého clusteru je okamžité díky automatické konfiguraci a připravené infrastruktuře v Docker Compose.
2. Migrace mezi prostředími je plně podporována díky použití volume svazků, které uchovávají persistentní data odděleně od kontejnerů.
3. Koncová bezpečnost systému je vysoká, protože všechny služby využívají interní autentizaci pomocí keyfile a žádný port kromě NGINX není vystaven veřejnosti.
4. Přidání nových uzlů do clusteru je rychlé a nevyžaduje zásah do běžící infrastruktury, stačí doplnit konfiguraci a spustit nové kontejnery.
5. Vytváření nových kolekcí se specifickými validačními schématy probíhá automaticky při startu systému, bez manuálních zásahů.
6. Ladění a sledování stavu systému je intuitivní díky přehlednému výstupu logů každé služby v konzoli.
7. Řešení je připraveno i pro produkční nasazení, protože obsahuje všechny klíčové komponenty MongoDB clusteru s důrazem na modularitu a bezpečnost.

***Nevýhody***

1. Úprava logiky inicializačních skriptů je náročná kvůli jejich složitosti a vzájemným závislostem mezi jednotlivými částmi.
2. Diagnostika chyb je obtížná v případě pádu některého z kontejnerů, protože jejich restart neprobíhá automaticky.
3. Spuštění Python skriptů pro naplnění databáze vyžaduje předchozí ruční instalaci knihoven na lokálním systému.
4. Údržba rozsáhlého clusteru je náročná na technické znalosti, protože nesprávná konfigurace shardingu nebo replikace může vést k nekonzistenci nebo ztrátě dat.

# Další specifika

Nejvýraznějším rozšířením je integrace NGINX jako reverzní proxy a load balanceru. Díky tomuto komponentu je možné směrovat veškerou komunikaci mezi klientem a MongoDB clusterem výhradně přes jeden veřejně dostupný port 27080. Veškeré požadavky jsou následně rovnoměrně distribuovány mezi dvě instance mongos, čímž je zajištěno:

* vyvážení zátěže při více současných připojeních,
* zvýšená dostupnost v případě výpadku jedné instance mongos,
* jasně definovaný a bezpečný vstupní bod do infrastruktury,
* možnost budoucí implementace HTTPS, rate limiting nebo ochrany pomocí firewallu.

Tato vrstva navíc umožňuje jednodušší škálování směrovací části systému, protože nové instance mongos mohou být přidány bez zásahu do konfigurace klientů.

# Data

V projektu jsou použity tři datové soubory ve formátu CSV, přičemž jeden z nich obsahuje více než 5 000 záznamů. Všechna data byla analyzována a transformována pomocí Python knihoven pandas, numpy a matplotlib, přičemž výstupy byly dokumentovány ve skriptu data\_analysis\_notebook.ipynb.

## Použité datové soubory

* **Netflix\_films.csv** – Datová sada filmů a seriálů ze streamovací služby Netflix. Obsahuje informace o názvu, žánru, hodnocení, zemi původu, roku vydání a dalších metadatech.
* **Indian\_Traffic\_Violations.csv** – Záznamy o dopravních přestupcích v Indii. Obsahují detailní informace o typu přestupku, věku řidiče, úrovni alkoholu, výši pokuty atd.
* **Mastercard\_stock\_history.csv** – Historické denní údaje o akciích společnosti Mastercard, včetně cen Open/High/Low/Close, objemu obchodů a dividend.

## Struktura a typy dat

Všechna data jsou nestrukturovaná nebo polosktrukturovaná, a proto byla zvolena dokumentová databáze MongoDB. Umožňuje efektivní práci s různorodými poli, včetně volitelných nebo částečně neúplných údajů. Formát CSV byl převeden do struktury BSON, která umožňuje snadnou indexaci i agregaci.

MongoDB je ideální pro tyto typy dat, protože nevyžaduje přesně definované schéma – lze snadno doplňovat nové klíče do záznamů bez nutnosti migrace celé databáze. Alternativní databázové modely jako grafové či sloupcové (např. Cassandra) by vyžadovaly jiný přístup ke strukturování těchto dat, a proto nebyly zvoleny.

## Rozsah dat

* **Netflix**: 8807 záznamů
* **Indian Traffic Violations**: 4,000 záznamů
* **Mastercard Stock History**: 5026 záznamů

Všechna data obsahují určitý počet prázdných nebo nulových hodnot – ty byly detekovány a popsány pomocí Pythonu. Například v sadě dopravních přestupků chybí u části záznamů informace o barvě vozidla nebo identifikaci policisty. Tyto hodnoty byly ponechány jako null.

## Úpravy dat

* **Konverze datových typů** – např. datum z formátu textového řetězce do datetime, čísla převedena z float na int tam, kde to bylo vhodné.
* **Odstranění záznamů s nulovou informativní hodnotou** – např. akcie s objemem 0 byly ze statistického zpracování vyřazeny.
* **Přidání nových polí** – během analýzy byla přidána např. pole legacy\_data nebo big\_dividend, která doplňují logiku systému.

## Zdroj dat

* Netflix: https://www.kaggle.com/datasets/shivamb/netflix-shows
* Indian Traffic Violations: https://www.kaggle.com/datasets/khushikyad001/indian-traffic-violation
* Mastercard stock history: https://www.kaggle.com/datasets/nelgiriyewithana/mastercard-historical-stock-data

## Analýza v Pythonu

Analýza probíhala v rámci Jupyter Notebooku data\_analysis\_notebook.ipynb, který obsahuje:

* počet záznamů, chybějících hodnot, unikátních hodnot,
* základní popisné statistiky (průměr, medián, rozptyl),
* histogramy a boxploty pro numerické hodnoty,
* korelace mezi proměnnými v sadách (např. vztah alkoholu a rychlosti),
* distribuci záznamů v čase (v případě akcií a přestupků).

Tento notebook slouží jako podklad pro ověření kvality a relevance dat v systému a je součástí odevzdávané dokumentace.

# Dotazy

Xxxxxx

Závěr

Xxxxxx

Zdroje

[1] Joshi, Priyam & Sarkar, Bikram. (2025). e-Commerce Website Using MERN Stack. DOI: 10.5281/zenodo.15102386. Dostupné z: <https://www.researchgate.net/publication/390272065_e-Commerce_Website_Using_MERN_Stack>

[2] IBM. What Is MongoDB? Dostupné z: <https://www.ibm.com/think/topics/mongodb>

[3] TechTarget. What is MongoDB? Features and how it works. Dostupné z: <https://www.techtarget.com/searchdatamanagement/definition/MongoDB>

Přílohy

Xxxxxx