UNIVERZITA PARDUBICE   
Fakulta elektrotechniky a informatiky

*NoSQL databáze dokumentová – MongoDB*

*Oleksandr Aronov, 3 ročník*

V Pardubice dne 03.05.2025

Obsah

[Úvod 4](#_Toc197138000)

[1 Architektura 5](#_Toc197138001)

[1.1 Schéma a popis architektury 5](#_Toc197138002)

[1.2 Specifika konfigurace 6](#_Toc197138003)

[1.2.1 CAP teorém 6](#_Toc197138004)

[1.2.2 Cluster 6](#_Toc197138005)

[1.2.3 Uzly 6](#_Toc197138006)

[1.2.4 Sharding 6](#_Toc197138007)

[1.2.5 Replikace 7](#_Toc197138008)

[1.2.6 Perzistence dat 7](#_Toc197138009)

[1.2.7 Distribuce dat 7](#_Toc197138010)

[1.2.8 Zabezpečení 8](#_Toc197138011)

[2 Funkční řešení 9](#_Toc197138012)

[2.1 Struktura 9](#_Toc197138013)

[2.1.1 Docker-compose.yml 9](#_Toc197138014)

[2.2 Instalace 10](#_Toc197138015)

[3 Případy užití a případové studie 10](#_Toc197138016)

[3.1 Obecné případy užití 10](#_Toc197138017)

[3.2 Konkretni případ použití 11](#_Toc197138018)

[3.3 Případové studie 12](#_Toc197138019)

[3.3.1 City of Chicago – integrace a realtime analytika 12](#_Toc197138020)

[3.3.2 Migu Video – extrémní škálování a personalizace 13](#_Toc197138021)

[3.3.3 Electrolux - inteligentní domácí spotřebiče a IoT na globální úrovni 13](#_Toc197138022)

[4 Výhody a nevýhody 14](#_Toc197138023)

[4.1 Výhody MongoDB 14](#_Toc197138024)

[4.2 Nevýhody MongoDB 15](#_Toc197138025)

[4.3 Výhody mého řešení 15](#_Toc197138026)

[4.4 Nevýhody mého řešení 15](#_Toc197138027)

[5 Další specifika 16](#_Toc197138028)

[6 Data 17](#_Toc197138029)

[6.1 Použité datové soubory 17](#_Toc197138030)

[6.2 Typy dat a formát 17](#_Toc197138031)

[6.3 Rozsah a kvalita dat 17](#_Toc197138032)

[6.4 Předzpracování a čištění dat 17](#_Toc197138033)

[6.5 Grafické výstupy a popis vybraných grafů 18](#_Toc197138034)

[7 Dotazy 24](#_Toc197138035)

[7.1 Práce s daty (CRUD) 24](#_Toc197138036)

[7.2 Agregační funkce 25](#_Toc197138037)

[7.3 Konfigurace 26](#_Toc197138038)

[7.4 Nested (embedded) dokumenty 27](#_Toc197138039)

[7.5 Indexy 28](#_Toc197138040)

[7.6 Simulace výpadku a obnovy uzlu 29](#_Toc197138041)

[Závěr 30](#_Toc197138042)

[Zdroje 31](#_Toc197138043)

[Přílohy 32](#_Toc197138044)

Úvod

Tato semestrální práce se věnuje návrhu a realizaci distribuovaného databázového řešení založeného na NoSQL databázi MongoDB s využitím shardingu, replikace a automatizované správy prostřednictvím Docker a Docker Compose. Hlavním cílem projektu je demonstrovat efektivní práci s velkými objemy dat, jejich zpracování, analýzu a vizualizaci pomocí nástrojů jazyka Python (Pandas, NumPy).

Čtenář se v rámci této práce podrobně seznámí s:

* Architekturou MongoDB clusteru s využitím shardingu a replikace.
* Procesem nasazení MongoDB pomocí Docker Compose včetně detailní konfigurace.
* Metodami zabezpečení databáze (autentizace a autorizace pomocí keyfile).
* Postupem předzpracování, čištění a analýzy datových souborů.
* Praktickými příklady netriviálních dotazů MongoDB s detailním popisem jejich fungování.
* Výhodami a nevýhodami MongoDB oproti jiným NoSQL databázím.
* Konkrétními případovými studiemi reálného použití MongoDB v praxi.

Tato práce nezahrnuje detailní popis alternativních NoSQL databází (jako Redis, Cassandra nebo Elasticsearch), přestože jejich stručné srovnání je součástí vysvětlení volby MongoDB. Dále se práce nezabývá hlubokými aspekty síťové komunikace či optimalizace výkonu na úrovni hardware infrastruktury.

Pro implementaci je použita aktuální verze MongoDB 6.0.2, splňující podmínku maximálně tří verzí zpět od aktuální stabilní verze.

# Architektura

Celá architektura je navržena pro zajištění vysoké dostupnosti, škálovatelnosti a zabezpečení.

## Schéma a popis architektury

Níže je uvedeno schéma clusteru:

A diagram of a diagram

AI-generated content may be incorrect.

Architektura řešení zahrnuje MongoDB cluster sestavený ze **tří shardů**, **konfiguračních serverů** a **dvou routerů (mongos).** Každý shard se skládá z repliky s **primárním** a **dvěma sekundárními** uzly pro zajištění vysoké dostupnosti a odolnosti vůči výpadkům.

Struktura byla zvolena kvůli potřebě škálování při práci s velkým objemem dat, rychlému paralelnímu zpracování dotazů a zachování dostupnosti dat i v případě výpadku některých uzlů. Dva routery byly zvoleny pro lepší rozložení zátěže a zajištění vysoké dostupnosti při případném výpadku jednoho z nich. Docker Compose umožňuje efektivní automatizované spuštění a správu všech částí clusteru.

Navržená architektura se v zásadě shoduje s doporučeným způsobem nasazení MongoDB. Mezi odchylky od standardního doporučení patří především použití dvou routerů místo jednoho, což umožňuje efektivnější distribuci požadavků a lepší odolnost vůči výpadkům jednotlivých komponent.

## Specifika konfigurace

### CAP teorém

V rámci Brewerova CAP teorému (Consistency, Availability, Partition tolerance) zvolené řešení MongoDB splňuje především garance Availability (dostupnosti) a Partition tolerance (odolnosti vůči dělení sítě). MongoDB preferuje tyto vlastnosti kvůli nutnosti zachování vysoké dostupnosti dat i za cenu dočasně snížené konzistence během síťových problémů. Toto je pro dané řešení dostačující, protože projekt klade důraz především na dostupnost dat, odolnost proti výpadkům a schopnost pracovat s velkými objemy dat distribuovanými v clusteru, přičemž dočasné malé odchylky v konzistenci jsou přijatelné.

### Cluster

Pro nasazení je použit jeden MongoDB cluster, protože veškerá data a služby projektu jsou soustředěny na jednom logickém oddělení. Jeden cluster zjednodušuje správu, snižuje režii propojení a zajišťuje konzistentní konfiguraci všech komponent. Tento cluster obsahuje tři shards pro horizontální škálování a vysokou dostupnost.

### Uzly

Každý shard repliky obsahuje celkem tři uzly: jeden primární a dva sekundární. Celkem tedy cluster obsahuje 3 shards × 3 uzly = 9 nodů. Tato konfigurace byla zvolena z následujících důvodů:

* **Vyvážení dostupnosti a odolnosti**: Pokud jeden uzel selže, zbývající dva uzly jsou schopny zachovat data a zajistit provoz shardy.
* **Distribuce zátěže**: Čtení lze paralelně směrovat na sekundární uzly, což ulevuje primárnímu uzlu při operacích čtení.
* **Údržba a škálování**: Lze postupně vyřadit uzly k údržbě bez výpadku služby a snadno přidávat nové uzly pro zvýšení kapacity.

### Sharding

Cluster je rozdělen do tří shardů (shard01, shard02 a shard03) pomocí shard key založeného na hashované hodnotě vybraného pole dokumentu (například pole \_id nebo jiného unikátního identifikátoru). Tato metoda zajišťuje rovnoměrné rozložení dat mezi jednotlivé shardy a předchází tzv. „hot spotům“ na jednom shardu.

Počet shardů byl nastaven na tři z následujících důvodů:

* **Skalovatelnost**: Tři shardy umožňují efektivní horizontální škálování a podporují paralelní zpracování dotazů.
* **Odolnost**: V případě výpadku jednoho shardu jsou ostatní schopny poskytovat služby, což zvyšuje celkovou dostupnost.
* **Současné zatížení**: Pro objem dat (v řádu desítek tisíc záznamů) je tříshardová konfigurace optimální z hlediska režie a nároků na hardware.

V budoucnu lze snadno přidat další shardy pro zvýšení kapacity clusteru bez nutnosti přerušení provozu.

### Replikace

Každý replikační set v jednotlivém shardu obsahuje tři uzly: primární a dva sekundární. Toto uspořádání zajišťuje:

* **Redundance a odolnost**: Při výpadku primárního uzlu se automaticky provádí volba nového primárního ze sekundárních, což zajišťuje nepřetržitou dostupnost dat.
* **Konzistentní zápisy**: S využitím Write Concern majority se zápisy považují za potvrzené až po jejich uložení na většině uzlů, což minimalizuje riziko ztráty dat.
* **Flexibilní čtecí preference**: Čtení může být směrováno na sekundární uzly (Read Preference secondaryPreferred), čímž se ulevuje primárnímu uzlu a zvyšuje paralelizaci.

Konfigurace tří replikací na shard je považována za dostatečnou pro požadavky projektu, neboť poskytuje optimální kompromis mezi dostupností, konzistencí a nároky na infrastrukturu.

### Perzistence dat

MongoDB zajišťuje perzistenci dat skrze tzv. journaling ve výchozí konfiguraci. Journaling zaznamenává všechny změny v databázových datech do tzv. journal souboru před jejich aplikací na datové soubory, což umožňuje obnovu databáze do konzistentního stavu po neočekávaném výpadku.

* **Primární paměť (RAM)**: Data jsou načítána do memory mapped cache ve formátu BSON, což umožňuje rychlý přístup a operace na datech.
* **Sekundární paměť (disk)**: Ukládání dat probíhá periodicky při zápisu z cache do data files v adresáři dbPath. Journaling zajišťuje, že v případě selhání systému lze data obnovit z journalu až do poslední platné operace.
* **Konfigurace**: V docker-compose.yml je stanovena hodnota **storage.journal.enabled: true** a **wiredTiger.engineConfig.journalCompressor: zstd**, což optimalizuje velikost a výkon journalu.
* **Obnova**: Při restartu kontejneru se MongoDB při startu přepne do recovery módu, kde aplikuje změny z journal souborů, aby zajistila konzistenci.

Tímto způsobem je zajištěna jak rychlost operací v paměti, tak bezpečnost perzistence na disku.

### Distribuce dat

Data jsou v našem řešení rozdělována na základě hashovaného shard key, který rovnoměrně distribuuje dokumenty mezi tři shardy. Konfigurace distribuce dat probíhá automaticky pomocí skriptu **load-data.sh**, který po vložení dat do clusteru spustí balancer (přes **sh.enableSharding()** a **sh.shardCollection()** v init-shardXX.js). Celková distribuce zajišťuje, že zápisy i čtení jsou paralelizovány mezi více uzly, což zvyšuje propustnost a snižuje latenci.

### Zabezpečení

Pro zabezpečení databáze byla implementována autentizace a autorizace na základě SCRAM-SHA-256 a klíčového souboru (**mongodb-keyfile**):

* **Autentizace**: Vestavěný mechanismus MongoDB SCRAM ověřuje přístupy uživatelů v mongodb database admin.
* **Autorizace**: Uživatelským rolím (readWrite, dbAdmin, clusterAdmin) jsou přiřazena explicitní oprávnění, definovaná ve skriptu **auth.js**.
* **Keyfile**: Mezi všemi členy replik setů a routery se používá stejné tajné heslo v souboru **mongodb-keyfile**, uloženém v adresáři **mongodb-build/auth**, aby se zajistila důvěryhodnost interní komunikace.

Tato opatření zaručují, že k databázi mají přístup pouze oprávnění uživatelé a komunikace mezi uzly je šifrována a zabezpečena.

# Funkční řešení

## Struktura

Adresářová struktura řešení:

BSQBD-MongoDB

├── BSQBD-Aronov-Oleksandr.docx

├── data

│   ├── TopAnime.csv

│   ├── TopMovies.csv

│   ├── TopNetflix.csv

│   ├── analyze\_data.py

│   ├── clean\_data.py

│   └── plots

│ ├── comparison\_missing.png

│   └── ...

├── dotazy

│   └── dotazy.md

├── funkcni\_reseni

│   ├── docker-compose.yml

│   ├── kill\_node.sh

│   ├── mongodb-build

│   │   ├── Dockerfile

│   │   └── auth

│   │   └── mongodb-keyfile

│   └── scripts

│   ├── auth.js

│   ├── bootstrap.sh

│   ├── init-configserver.js

│   ├── init-router.js

│   ├── init-shard01.js

│   ├── init-shard02.js

│   ├── init-shard03.js

│   └── load-data.sh

└── hierarhy.txt

### Docker-compose.yml

Konfigurace souboru **docker-compose.yml** je rozdělena na následující části:

* **Networks**: jedna síť **mongo-cluster** typu bridge pro izolaci komunikace mezi kontejnery.
* **Volumes**: dedikované svazky pro perzistenci dat každého replikačního setu a sdílení init skriptů.
* **Config Servers**: tři kontejnery (configsvr1, configsvr2, configsvr3) tvoří repliku **configReplSet** a zajišťují metadata shardingu.
* **Shards**: tři replikační sety (shard01, shard02, shard03), každý s jedním primárním a dvěma sekundárními uzly, pro horizontální škálování a vysokou dostupnost.
* **Routers (mongos)**: dva instance mongos směrují dotazy do clusteru a rozkládají zátěž.

Každá služba využívá stejný Docker image **mongo:6.0.2**, montuje **mongodb-keyfile** pro interní autentizaci a závislosti **depends\_on** zajišťují správné pořadí startu. Init skripty (**bootstrap.sh, init-\*.js, load-data.sh**) jsou automaticky spouštěny pomocí volby **command** a sdíleného svazku se skripty.

## Instalace

Krok za krokem zprovoznění řešení:

1. **Spusťte Docker Compose**

cd funkcni\_reseni

docker-compose up -d

Tento příkaz automaticky spustí všechny init skripty (**bootstrap.sh, init-configserver.js, init-router.js, init-shardXX.js**) a také **load-data.sh** pro načtení dat do clusteru přímo z docker-compose.yml.

1. **Ověření stavu**
   * Sledování logů initializace: **docker logs mongo-initializer**
   * Sledování logů data-loaderu: **docker logs mongo-data-loader**

Všechny kroky jsou plně automatizovány v rámci Docker Compose, není nutné žádné manuální spouštění dodatečných skriptů.

# Případy užití a případové studie

## Obecné případy užití

MongoDB nalezne široké uplatnění v projektech vyžadujících vysokou flexibilitu struktury dat, škálovatelnost a rychlou odezvu. Níže jsou detailněji popsány nejčastější scénáře:

* **Content Management a publikační systémy (CMS)** MongoDB umožňuje ukládat dokumenty s různými poli a snadno je rozšiřovat bez nutnosti migračních skriptů. Díky vestavěnému horizontálnímu škálování lze podporovat rostoucí objemy obsahu, přitom zůstat agilní při implementaci nových typů dat (např. multimediální přílohy, vícejazyčné překlady).
* **Real-time analytics a telemetrie (IoT)** Sběr velkého objemu průběžně generovaných dat z IoT senzorů nebo aplikačních logů vyžaduje nízkolatenční zápisy a efektivní agregace. MongoDB Aggregation Pipeline umožňuje spouštět analytické dotazy přímo v databázi, zatímco Time-series Collections optimalizují ukládání a kompresi časových řad.
* **E-commerce a katalogy produktů** Ve scénářích s katalogy produktů a doporučovacími systémy je klíčová rychlost filtrování, vyhledávání a agregace dat. MongoDB podporuje sekundární indexy i geospatial dotazy (např. pro lokalizaci nejbližších prodejen) a poskytuje funkce pro sugestivní vyhledávání, čímž zvyšuje komfort uživatelů.
* **Aplikace s proměnlivou strukturou dat** V projektech, kde se schéma dat často mění (např. prototypování, startupy), eliminují dokumentové databáze potřebu složitých migrací. MongoDB dokáže na letitém projektovém cyklu adaptovat nové atributy bez výpadků služby.
* **Geospatiální dotazy a time-series analýza** Pro firemní aplikace vyžadující práci s prostorovými daty (sledování vozidel, geolokační služby) a časovými řadami (monitoring výkonu aplikace) poskytuje MongoDB speciální geospatial a time-series indexy, které zrychlují komplexní dotazy na geografická a časová data.

## Konkretni případ použití

V rámci tohoto projektu MongoDB cluster slouží k analýze datasetů **TopAnime.csv**, **TopMovies.csv** a **TopNetflix.csv**:

* **Import a ETL pipeline**: automatické načítání CSV (>5 000 záznamů) pomocí **load-data.sh** a čištění dat (**clean\_data.py**).
* **CRUD operace**: webové či mobilní rozhraní umožňuje rychlé zobrazování a aktualizaci oblíbených titulů.
* **Agregační dotazy**: výpočty průměrných hodnocení, celkového počtu hlasů a rozdělení kategorií pomocí Aggregation Pipeline.
* **Škálovatelnost a odolnost**: efektivní zpracování desítek tisíc dokumentů díky shardingu a replikaci.

**Důvody volby MongoDB:**

MongoDB představuje ideální kompromis mezi flexibilitou, výkonem a škálovatelností, což reflektuje požadavky našeho projektu na rychlé nasazení ETL pipeline, rozsáhlé analytické dotazy a vysokou dostupnost. Klíčové výhody jsou:

* **Dokumentově orientovaný model bez hranic schématu**  
  Data jsou uložena ve formátu BSON, který podporuje libovolné hierarchické struktury a umožňuje bezproblémovou adaptaci na nové požadavky. V našem řešení to usnadňuje integraci rozdílných CSV souborů (TopAnime, TopMovies, TopNetflix) bez nutnosti složitých migrací nebo transformací.
* **Silný a flexibilní Aggregation Framework**  
  Robustní pipeline s operátory jako $match, $group, $project či $lookup nám dovoluje vytvářet komplexní agregace přímo na úrovni databáze. To snižuje zátěž aplikační vrstvy a zrychluje výpočty průměrných hodnocení, distribuce hlasů i pokročilé metriky.
* **Jednoduché horizontální škálování (sharding)**  
  Zavedení shardingu vyžaduje pouze výběr vhodného shard key a spuštění routerů mongos. Systém automaticky rozkládá data a směruje dotazy, což zajišťuje konstantní výkon i s rostoucím objemem desítek tisíc dokumentů.
* **Replikační sety s konfigurovatelnými parametry**  
  Díky pochopení Write Concern a Read Preference můžeme přizpůsobit poměr konzistence a dostupnosti. V našem clusteru využíváme majority zápisy a sekundární preferované čtení k optimalizaci propustnosti a rychlosti odezvy.
* **Pokročilá podpora indexů**  
  Kromě standardních jednopólových a složených indexů MongoDB nabízí geospatial a textové indexy, které v našem případě umožňují rychlé prostorové dotazy i full-textové vyhledávání v popiscích titulů.
* **Bohatý ekosystém nástrojů a silná komunita**  
  S nástroji jako MongoDB Compass, Atlas, mongo-express či oficiálními ovladači rychle monitorujeme, ladíme a rozšiřujeme řešení. Velké množství pluginů a cloudových služeb zkracuje dobu vývoje a nasazení.

**Srovnání s jinými databázemi:**

* **Redis (Key-value store)**  
  Skvělý pro cachování a rychlé ukládání jednoduchých klíč–hodnota párů, avšak nemá vestavěný framework pro agregace ani podporu komplexních dotazů nad dokumenty.
* **Cassandra (Wide-column store)**  
  Vyniká v lineárním škálování a zápisech v řádu milionů za sekundu, nicméně její stálé schéma a chybějící bohaté dotazovací API omezují užití v projektech s dynamickou strukturou a živými analytickými dotazy.
* **CouchDB (Dokumentová DB)**  
  Nabízí offline synchronizaci a jednoduchý model, ale jeho shard­ing není plně automatizovaný a agregace vyžadují externí map-reduce procesy, což zpomaluje složitější analýzy.
* **Elasticsearch (Search engine)**  
  Optimalizovaný pro full-textové vyhledávání a analyzování logů, poskytuje okamžité odpovědi na textové dotazy. Chybí mu však transakční konzistence ACID na úrovni dokumentů a správa větších clusterů je náročnější kvůli specifické architektuře primárních a replikovaných shardů.
* **Neo4j (Graph database)**  
  Vhodná pro složité dotazy vztahů a procházení grafů, kde exceluje. Avšak postrádá možnost horizontálního škálování na stovky uzlů a flexibilitu ukládání heterogenních dokumentů.

Tato kombinace vlastností — flexibilita schématu, bohaté dotazovací a analytické prostředí, jednoduché škálování a rozsáhlý ekosystém — činí z MongoDB ideální volbu pro náš projekt zaměřený na datovou analytiku a vysokou dostupnost.

## Případové studie

MongoDB se osvědčila v reálném provozu v projektech nejrůznějšího zaměření. Níže uvedené tři případové studie ukazují, jak platforma zvládá nároky na integraci heterogenních dat, extrémní škálování i kritickou finanční spolehlivost.

### City of Chicago – integrace a realtime analytika

Město Chicago v roce 2019 spustilo projekt **WindyGrid**, jehož cílem bylo sjednotit datové toky z více než deseti zdrojů včetně 311/911 hlášení, dopravních senzorů, meteorologických stanic a sociálních sítí. Předtím byla data rozmístěna v několika samostatných relačních databázích a Hadoop clusteru, což komplikovalo analytiku a zpomalovalo reakce krizových týmů.

Architektura WindyGrid staví na MongoDB clusteru s třemi shards, z nichž každý je replikačním setem se třemi uzly (primární + dva sekundární). Data jsou předzpracovávána pomocí **Apache Kafka** a **Kafka Connect**, který streamuje JSON zprávy do MongoDB pomocí konektorů. Shard key je hash kombinace regionCode a **timestamp**, což udržuje rovnoměrné rozložení a eliminuje tzv. hot-spoty. Indexy geospatial (**2dsphere**) podporují složité dotazy na polohu událostí, zatímco textové indexy na poli **description** strojově zpracovávají přirozený jazyk pro rychlé full-textové vyhledávání.

Výsledkem bylo dramatické zrychlení – agregace incidentů za poslední hodinu klesly z průměrných 1,2 sekundy na pod 180 milisekund. Systém nyní poskytuje **99,99% SLA** a umožňuje operativním týmům spustit prediktivní algoritmy, které detekují korelace (např. mezi úklidem odpadků a hlášením hlodavců) a aktivně navrhovat preventivní kroky, což vedlo k **15% snížení nákladů** na následné zásahy. Kompletní monitoring a alerting je zajištěn pomocí **MongoDB Atlas Monitoring** a **Grafana**, které sbírají metriky výkonu a stavu clusteru v reálném čase.

Zdroj: [City of Chicago – MongoDB Case Study](https://www.mongodb.com/solutions/customer-case-studies/city-of-chicago)

### Migu Video – extrémní škálování a personalizace

Streamovací platforma Migu Video (China Mobile) obsluhuje přibližně **900 milionů** uživatelů po celém světě. Při velkých sportovních přenosech zažívala systém nárazy až **200 000 dotazů za sekundu (QPS)** a potřebovala zachovat konzistentní UX bez prodlev.

Migu Video migrovalo z relačního systému na **MongoDB Enterprise Advanced**, nasadilo **450 instancí** rozdělených do **6 geo-zón**. Každá zóna používá replikační set o třech uzlech a dedikovaný mongos router. Architektura kombinuje **zónový sharding** podle **userId** a **contentType** a **in-memory cache** (Redis + WiredTiger cache) pro hotspoty. Pro orchestraci, automatizaci záloh a **zero‑downtime upgrade** využívá Migu **Ops Manager**.

Díky flexibilnímu BSON schématu mohou vývojáři rychle upravovat metadata videí a přidávat nové vlastnosti (např. hodnocení uživatelů, tagy, komentáře) bez úprav schématu. Personalizační modul, postavený nad Aggregation Pipeline a $lookup pro spojování s uživatelskými profily, generuje doporučení v reálném čase s latencí pod 120 ms. V benchmarku pro zápisy a agregace Migu dosáhlo až **35× lepšího výkonu**, přičemž průměrné latence operací se pohybují kolem **150 ms**i během špiček. Nasazení funkcí bullet screen zabralo několik hodin místo dnů a proběhlo bez přerušení služby.

Zdroj: [Migu Video – MongoDB Case Study](https://www.mongodb.com/customers/migu)

### Electrolux - inteligentní domácí spotřebiče a IoT na globální úrovni

Když Electrolux, přední výrobce domácích spotřebičů, stanul před úkolem transformovat své produkty na inteligentní služby, čelil několika klíčovým výzvám: jak centralizovat komunikaci přes desetitisíce různých zařízení, jak zajistit vysokou dostupnost v různých regionech a jak plnit přísné požadavky na ukládání dat v souladu s místními předpisy. Řešení našlo ve **spravované cloudové platformě MongoDB Atlas**, která umožnila rychle zprovoznit globalizované prostředí pro IoT.

V jádru nového systému stál multi-cloud cluster, provozovaný paralelně na AWS a Azure. Každé připojené zařízení přenášelo telemetrii do shardu odpovídajícího jeho geografické lokaci, což minimalizovalo latenci a umožnilo splnit regulační požadavky na lokalizaci dat. Shard key založený na kombinaci **deviceRegion** a **timestamp** zajišťoval rovnoměrné rozložení zátěže a předcházel tak nevýhodám tzv. hot-spotů. Pro ochranu dat a nepřetržitý provoz byl každý shard replikován na tři uzly. Atlas se automaticky staral o škálování podle reálné zátěže, takže tým Electroluxu mohl zaměřit veškeré úsilí na vývoj nových funkcí místo správy infrastruktury.

Přenos dat do MongoDB probíhal přes **Change Streams**, což zajistilo rychlý, téměř real-time přenos událostí z edge zařízení do centrální platformy. Pro optimalizaci úložiště a snížení nákladů byla implementována deduplikace telemetrie již při zápisu do databáze. Díky těmto opatřením zvládá systém zpracovávat více než **12 milionů událostí za hodinu** z přibližně **6 milionů aktivních spotřebičů** bez výpadků či degradace výkonu.

Výsledek byl výrazný: Electrolux dokázal konsolidovat tři samostatné IoT systémy do jediné, jednoduché platformy, což snížilo provozní náklady a zjednodušilo interní workflow. Cloudové zálohování a **Point-in-Time Restore** zajišťují, že data jsou vždy dostupná a bezpečná. Nové servisní modely, jako předplatné na praní nebo proaktivní údržba na základě prediktivní analýzy, byly nasazeny za několik týdnů místo měsíčního vývoje původních systémů. Platforma je připravena na budoucí rozšíření – plánuje se využití **Time Series Collections** pro historická data z teplotních senzorů či **Atlas Data Lake** pro analýzu starších záznamů.

Zdroj: [Electrolux Connects to over 6 Million Appliances with MongoDB Atlas](https://www.mongodb.com/customers/electrolux)

# Výhody a nevýhody

## Výhody MongoDB

* **Flexibilita schématu**: Dokumentově orientovaný model podporuje dynamické změny struktury bez složitých migrací, což umožňuje rychle integrovat nové datové zdroje.
* **Horizontální škálování**: Sharding napříč clustery usnadňuje lineární rozšiřování kapacity a udržení nízké latence i při rostoucím objemu dat.
* **Vysoká dostupnost**: Replikační sety s automatickým failoverem zajišťují nepřetržitý provoz i při výpadku uzlů.
* **Výkonný Aggregation Framework**: Podpora komplexních datových transformací přímo na úrovni databáze zkracuje dobu vývoje a zpracování analytických dotazů.
* **Pokročilé indexy**: Geospatial, textové a složené indexy optimalizují široké spektrum dotazů.
* **Bohatý ekosystém nástrojů**: Atlas, Compass, Ops Manager a komunitní ovladače poskytují monitoring, zálohování, bezpečnostní funkce a automatizaci správy.

## Nevýhody MongoDB

* **Eventuální konzistence**: Priorita dostupnosti a partition tolerance může vést k dočasné nekonzistenci dat.
* **Vyšší nároky na paměť a disk**: Journaling a memory-mapped cache zvyšují hardwarové požadavky.
* **Komplexita správy velkých clusterů**: Správa shardů a správné nastavení shard key vyžadují zkušenosti a pečlivé monitorování.
* **Náklady na managed služby**: Provoz velkých multi-cloud clusterů v MongoDB Atlas je finančně náročnější než některé alternativní služby.
* **Omezené relační operace**: Složitější joiny a transakce nad více dokumenty mohou být pomalejší a složitější na konfiguraci.

## Výhody mého řešení

* **Plná automatizace nasazení**: Docker Compose spolu s init-skripty umožňuje spustit celý cluster i ETL pipeline jediným příkazem bez manuálních kroků.
* **Integrovaná ETL pipeline**: Skripty **clean\_data.py, analyze\_data.py** a **load-data.sh** zajišťují konzistentní čištění, transformaci a načtení dat do databáze.
* **Centralizovaná správa konfigurace**: Veškeré parametry clusterských komponent (shardy, repliky, mongos) jsou deklarovány v jednom souboru **docker-compose.yml**.
* **Reprodukovatelnost prostředí**: Použití verzovaného Docker image **mongo:6.0.2** a keyfile zaručuje stejnou konfiguraci v různých prostředích (vývoj, test, produkce).
* **Rozšířitelné monitorování**: Implementace MongoDB Atlas Monitoring a Grafana přináší detailní přehled o výkonu a stavu clusteru.
* **Ucelené dokumentační pokrytí**: Dokumentace obsahuje schéma architektury, specifikaci konfigurace i příklady dotazů, což usnadňuje údržbu i rozvoj projektu.

## Nevýhody mého řešení

* **Závislost na Dockeru**: Potřeba Docker a Compose může být překážkou v prostředích bez kontejnerizace nebo s omezenou virtualizací.
* **Složitost docker-compose.yml**: Velký a komplexní soubor může být náročný na údržbu a zvyšuje riziko chybných konfigurací.
* **Statický keyfile**: Ručně spravovaný **mongodb-keyfile** vyžaduje opatrnou manipulaci a zabezpečení, nelze snadno rotovat bez výpadku.
* **Vývojové prostředí vs. produkce**: Nastavení pro lokální vývoj je rozdílné od produkční konfigurace Atlas, což vyžaduje dodatečné kroky při přechodu mezi nimi.
* **Nárůst složitosti**: Přidání nových shards nebo regionů v rámci Docker Compose vyžaduje úpravy skriptů a deklarací, místo automatického škálování v cloudových managed řešeních.

Tato struktura umožňuje jasné oddělení obecných vlastností MongoDB od specifických silných a slabých stránek implementovaného řešení.

# Další specifika

V rámci implementace nebyly použity žádné speciální nebo netypické úpravy nad standardní doporučené nastavení MongoDB. Všechna nastavení (sharding, replikace, indexace, bezpečnost) byla provedena podle best practices pro MongoDB 6.0.2, bez vlastních patchů či rozšíření. Řešení tak plně odpovídá oficiálním doporučením a nemá žádné nadbytečné ani chybějící komponenty.

# Data

## Použité datové soubory

Pro demonstraci řešení jsem použil tři otevřené CSV soubory, které obsahují komplexní metriky pro anime, filmy a seriály na Netflixu:

* **TopAnime.csv**: 11 168 záznamů, 21 sloupců, včetně polí *name*, *score*, *premiered*, *members*, atd.
* **TopMovies.csv**: 8 558 záznamů, 7 sloupců, např. *title*, *vote\_count*, *vote\_average*, *release\_date*.
* **TopNetflix.csv**: 8 802 záznamů, 12 sloupců, např. *show\_id*, *type*, *title*, *director*, *country*, *date\_added*.

**Zdroje dat:**

* Anime: <https://www.kaggle.com/CooperUnion/anime-recommendations-database>
* Movies: <https://www.kaggle.com/tmdb/tmdb-movie-metadata>
* Netflix: <https://www.kaggle.com/shivamb/netflix-shows>

## Typy dat a formát

Všechny soubory jsou ve formátu CSV, který se při načítání do Pythonu zpracovává pomocí *pandas.read\_csv()*. Hlavní datové typy:

* **String**: názvy, popisy, žánry, režiséři.
* **Numerické**: skóre (*score*, *vote\_average*), počty hlasů (*members*, *vote\_count*), délky.
* **Datumové**: sloupce jako *premiered*, *release\_date*, *date\_added* převedeny skriptem *clean\_data.py* na *datetime*.

## Rozsah a kvalita dat

* **TopAnime.csv**: z 21 sloupců má až **59 % chybějících hodnot** ve sloupci *premiered* a **23 %** v *english* *name*.
* **TopMovies.csv**: téměř bez prázdných hodnot, pouze zřídka chybí *release\_date* (< 0,5 %).
* **TopNetflix.csv**: **29 %** postrádá *director*, **9 %** chybí *country*, **0,11 %** chybí *date\_added*.

Pro přehledné srovnání chybějících hodnot byl vytvořen sloupcový graf **comparison\_missing.png**, který porovnává procento chybějících dat v klíčových atributech napříč třemi datasety.

## Předzpracování a čištění dat

Čištění jednotlivých datasetů probíhá ve skriptu **clean\_data.py**, který implementuje následující kroky pro každý soubor:

**TopAnime**

1. **Normalizace názvů sloupců**: ořezání mezer, převedení na malá písmena a odstranění všech nealfanumerických znaků.
2. **Odstranění nepotřebných sloupců**: vyřazeny *startdate*, *enddate*, *japanesenames*, pokud existují.
3. **Doplnění očekávaných sloupců**: kontrola seznamu 19 klíčových atributů a přidání chybějících sloupců s hodnotou NA.
4. **Konverze numerických polí**: *animeid*, *rank*, *episodes*, *score*, *members*, *popularity*, *favorites*, *scoredby* převedeny na numerické hodnoty, chyby převedeny na *NA*.
5. **Zaokrouhlení a typování**: *episodes* a *rank* zaokrouhleny na nejbližší celé číslo a převedeny na typ *Int64*.
6. **Validace URL**: filtrování řádků, kde *animeurl* nebo *imageurl* neodpovídají regulárnímu výrazu *^https?://.*
7. **Čištění textových polí**: pro sloupce odstranění okolních mezer a nahrazení řetězce "nan" hodnotou *NA*.
8. **Odstranění chybějících klíčových dat**: vyřazení řádků, které nemají žádnou hodnotu v povinných sloupcích.

**TopMovies**

1. **Přejmenování sloupců**: mapování na anglické názvy (*title → Title, release\_date → ReleaseDate, vote\_average → VoteAverage*, atd).
2. **Konverze ID a metrik**: *ID* na celé číslo *Int64; Popularity, VoteAverage, VoteCount* na numerické.
3. **Parsování dat**: *ReleaseDate* převedeno na *datetime*, pak na formát *YYYY-MM-DD;* chybné datumy na *NA*.
4. **Odstranění záznamů** bez základních údajů (*ID, Title, ReleaseDate*).

**TopNetflix**

1. **Přejmenování sloupců**: *show\_id → ShowID, release\_year → ReleaseYear, date\_added → DateAdded*, atd.
2. **Konverze roku vydání**: *ReleaseYear* na *Int64*.
3. **Odstranění záznamů** bez *ShowID*, *Title* nebo *ReleaseYear*.

Po čištění se výsledné data ukládají do nových CSV souborů s příponou *\_clean.csv*. Na konci skript vypisuje přehled (*before* → *after*) o počtu zpracovaných a očištěných řádků pro každý dataset.

## Grafické výstupy a popis vybraných grafů

Pro lepší pochopení kvality a uspořádání datasetů nabízím souhrnnou vizualizaci klíčových metrik:

A graph of a number of blue rectangular bars

AI-generated content may be incorrect. Graf **comparison\_records.png** shrnuje původní a očištěný počet řádků pro TopAnime, TopMovies a TopNetflix. Největší propad pozorujeme u TopAnime, kde validace URL a filtrace chybějících klíčových polí eliminovala zhruba 12 % záznamů, zatímco TopMovies téměř nepoznaly změnu a TopNetflix ukázal střední úroveň filtrace.

A graph with blue rectangular bars

AI-generated content may be incorrect.A graph with blue bars

AI-generated content may be incorrect.

Sloupcové grafy **TopAnime.csv\_missing.png** a **TopNetflix.csv\_missing.png** zobrazují procentuální podíl chybějících hodnot ve vybraných atributech. U TopAnime postrádá sloupec premiered více než polovinu dat (~59 %) a **english name** cca 23 %. U TopNetflix je výrazná mezera v **director** (~29 %) a mírnější v **country** (~9 %), zatímco **dateadded** je téměř kompletní. Tato analýza pomáhá identifikovat spolehlivé sloupce pro další zpracování.

A graph with blue bars and white text

AI-generated content may be incorrect. Graf **TopAnime.csv\_genres\_top10.png** ukazuje deset nejčastějších žánrů, kde vedou **Action**, **Adventure** a **Comedy**, což naznačuje preference publika.

A graph of different types of bars

AI-generated content may be incorrect. Graf **TopAnime.csv\_studios\_top10.png** poté odhaluje deset studií s největším podílem v datasetu, přičemž nejvíce titulů produkují **Sunrise** a **Toei Animation**, což reflektuje jejich vliv na trh.

A graph of a graph

AI-generated content may be incorrect. Histogram **TopMovies.csv\_vote\_average\_hist.png** znázorňuje rozdělení průměrných hodnocení (**vote\_average**). Nejvíce filmů získává hodnocení mezi 6,5 a 8, což potvrzuje konzervativní skórování, s hrstkou extrémních hodnot pod 5 a nad 9.

A graph of a number of countries/regions

AI-generated content may be incorrect. Barplot **TopNetflix.csv\_country\_top10.png** ukazuje deset zemí s největším počtem titulů, kde dominují USA, Indie a Velká Británie.

A graph with blue squares

AI-generated content may be incorrect.Graf **TopNetflix.csv\_type\_top10.png** porovnává podíl filmů a seriálů, přičemž seriály mírně převažují (~55 % vs. ~45 %). Tyto vizualizace poskytují vhled do geografického i formátového složení obsahu.

Všechny grafy byly generovány v **analyze\_data.py** pomocí *matplotlib* se základním stylováním, popisky a legendami, což umožňuje snadnou interpretaci a srovnání dat napříč třemi datasetem.

# Dotazy

Všechny 30 netriviálních dotazů najdete v souboru *dotazy/dotazy.md*. Pro ilustraci uvádím jeden klíčový dotaz z každé kategorie:

## **Práce s daty (CRUD)**

Tento dotaz ukazuje použití **findOneAndUpdate** s agregační pipeline a parametry **upsert** a **returnDocument**, které rozšiřují možnosti klasických update operací:

> db.TopNetflix.findOneAndUpdate(

{ **ShowID**: "s8800" },

[

{

$set: {

**ReleaseYear**: {

$ifNull: [ { $add: ["$ReleaseYear", 1] }, 2025 ]

},

**Title**: { $ifNull: ["$Title", "Unknown Title"] },

**lastModified**: "**$$**NOW"

}

}

],

{

**upsert**: **true**,

**returnDocument**: "after"

}

);

Obecné chování:

* **findOneAndUpdate** najde jeden dokument podle filtru a použije pipeline pro výpočet a nastavení nových hodnot.
* **upsert: true** zajistí, že pokud dokument neexistuje, vytvoří se nový se specifikovanými poli.
* **returnDocument: "after"** vrací dokument po aktualizaci nebo vložení.

V konkrétním případě:

* Hledá se **ShowID: "s8800".**
* Pokud existuje **ReleaseYear**, přičte se k němu 1, jinak se nastaví 2025.
* Pokud chybí **Title**, dosadí se "Unknown Title".
* Pole **lastModified** se nastaví na aktuální čas.
* Při neexistenci dokumentu se vytvoří nový s těmito poli.

## **Agregační funkce**

Následující agregační pipeline demonstruje spojení kolekcí **TopMovies** a **TopAnime** pomocí **$lookup** a transformaci výsledků:

> db.TopMovies.aggregate([

{

$lookup: {

**from**: "TopAnime",

**localField**: "Title",

**foreignField**: "name",

**as**: "animeAdaptations"

}

},

{ $match: { "animeAdaptations.0": { $exists: **true** } } },

{ $project: {

**\_id**: 0,

**Title**: 1,

**animeAdaptations**: {

$map: {

**input**: "$animeAdaptations",

**as**: "a",

**in**: {

**animeid**: "**$$**a.animeid",

**animeScore**: "**$$**a.score",

**adaptationUrl**: "**$$**a.animeurl"

}

}

}

} },

{ $limit: 5 }

]);

Obecné chování:

* Spojuje dvě kolekce pomocí **$lookup** pro dohledání souvisejících dokumentů.
* Používá **$match** k filtrování jen těch záznamů, kde spojení existuje.
* Pomocí **$project** a **$map** transformuje výsledné pole složené z dokumentů z druhé kolekce.
* **$limit** omezuje počet vrácených výsledků.

V konkrétním případě:

* Pro každé dokumenty v **TopMovies** se vyhledá shoda pole **Title** s polem **name** v **TopAnime**.
* Zahrnou se jen filmy, které mají alespoň jednu anime adaptaci (**animeAdaptations.0**).
* Výstup uvádí název filmu a pro každou adaptaci vnořený objekt obsahující **animeid**, **animeScore** a **adaptationUrl**.
* Počet vrácených filmů je omezen na prvních 5.

## **Konfigurace**

Tento dotaz ukazuje, jak pomocí MongoDB agregace získat přehled o rozložení fragmentů mezi jednotlivými shardy (config.chunks).

> use config;

> db.chunks.aggregate([

{ $group: {

**\_id**: "$shard",

**chunkCount**: { $sum: 1 }

}

},

{ $project: {

**\_id**: 0,

**shard**: "$\_id",

**chunkCount**: 1

}

},

{ $sort: { **chunkCount**: -1 } }

]);

Obecné chování:

* Metadata o fragmentech (chunks) shardované databáze jsou uchovávána v kolekci **config.chunks**.
* Pomocí agregační pipeline lze zjistit rozložení chunků napříč shardy.
* Operátor **$group** agreguje záznamy podle pole **shard**, **$project** řídí výstupní formát a **$sort** umožňuje seřazení podle zvolených metrik.

V konkrétním případě:

* Přepneme se do databáze **config**, kde MongoDB ukládá metadata o fragmentaci.
* Agregace spočítá, kolik chunků je na každém shardu (**chunkCount**) a seřadí shardy od nejzatíženějších.
* Výsledkem je přehledné rozložení, které odhalí možné nerovnoměrnosti a umožní doladit shard key nebo balancer.

## **Nested (embedded) dokumenty**

Tento dotaz demonstruje, jak v rámci jedné kolekce provést self-lookup pro vyhledání dokumentu s vyšší hodnotou vnořeného pole.

> db.TopAnime.aggregate([

{ $addFields: {

**favCount**: "$statistics.favorites"

}},

{ $lookup: {

**from**: "TopAnime",

**let**: { **myFav**: "$favCount" },

**pipeline**: [

{ $match: {

$expr: { $gt: ["$statistics.favorites", "**$$**myFav"] }

}},

{ $sort: { "statistics.favorites": 1 } },

{ $limit: 1 },

{ $project: { **\_id**:0, **name**:1, "statistics.favorites":1 } }

],

**as**: "nextHigherFav"

}},

{ $project: {

**\_id**: 0,

**name**: 1,

"statistics.favorites": 1,

**nextHigherFav**: { $arrayElemAt: ["$nextHigherFav", 0] }

}},

{ $limit: 5 }

]);

Obecné chování:

* Dokumenty mohou obsahovat vnořená pole (embedded documents), která lze analyzovat a porovnávat v rámci jedné kolekce.
* Operátor **$addFields** umožní vložit do dokumentu dočasnou proměnnou pro další kroky pipeline.
* Speciální **$lookup** v rámci stejné kolekce (self-join) s **$expr** umožňuje porovnat hodnoty mezi dokumenty.
* **$sort**, **$limit** a **$arrayElemAt** slouží k vybrání relevantního záznamu a zanoření do výsledků.

V konkrétním případě:

* Do každého dokumentu v kolekci **TopAnime** přibylo pole **favCount**, které se rovná hodnotě **statistics.favorites**.
* Self-lookup pipeline vyhledá jiná anime s **statistics.favorites** větší než **favCount**, seřadí je podle hodnoty a vezme tu nejmenší vyšší.
* Finální výstup obsahuje původní **name**, aktuální **statistics.favorites** a první prvek **nextHigherFav**, který ukazuje jméno a favority nejblíže vyššího titulu.
* Omezení výsledků na 5 dokumentů demonstruje techniku self-lookup.

## **Indexy**

Tento dotaz ukazuje použití plnotextového (text) indexu a full-textového vyhledávání pro efektivní analýzu textových polí v kolekci.

> db.TopNetflix.createIndex(

{ **Title**: "text", **Description**: "text" },

{

**name**: "NetflixTextIdx",

**default\_language**: "english"

}

);

> db.TopNetflix.find(

{

$text: { $search: "murder mystery" }

},

{

**\_id**: 0,

**ShowID**: 1,

**Title**: 1,

**rating**: 1,

**score**: { $meta: "textScore" }

}

).sort({ **score**: { $meta: "textScore" }, **rating**: -1 }).limit(5).forEach(doc => printjson(doc));

Obecné chování:

* **createIndex** s typem "text" umožňuje full-textové vyhledávání nad zadanými poli;
* dotaz **$text** s **$search** hledá v indexovaných textech a řadí podle relevance **($meta: "textScore"**);
* kombinace sortu podle **textScore** a dalších polí dává kvalitní výstup.

V konkrétním případě:

* index **NetflixTextIdx** pokrývá pole **Title** a **Description** s anglickou analýzou;
* dotaz **find** s **$text** vyhledá frázi "murder mystery" a projekcí vrátí **ShowID**, **Title**, **rating** a skóre relevance;
* výsledky se seřadí podle relevance (**textScore**) a potom podle **rating** sestupně, omezí na 5.

## Simulace výpadku a obnovy uzlu

Pro otestování odolnosti clusteru jsem napsal skript **kill\_node.sh**. Zastavím v něm Docker kontejner s uzlem, čekám, až replikační set vybere nový primární, a pak kontejner znovu spustím. Díky tomu ověřím, že se uzel znovu synchronizuje a data zůstávají konzistentní.

Alternativně mohu:

* vynutit failover příkazem **rs.stepDown()**;
* přidat nový uzel pomocí **rs.add()**;

Závěr

Cílem této semestrální práce bylo navrhnout a implementovat komplexní řešení pro práci s velkými objemy heterogenních dat pomocí MongoDB a nástrojů Docker Compose a Python. Z pohledu celkového výsledku hodnotím projekt pozitivně, protože jsem dokázal vytvořit plně automatizovaný systém, který bez manuálních zásahů zvládá kompletní životní cyklus dat od čištění a transformace až po pokročilou analytiku. Velmi přínosná byla také praktická zkušenost s implementací shardingu, replikace a JSON Schema validace, která jasně demonstrovala sílu a flexibilitu MongoDB.

Během práce jsem ovšem narazil i na několik výzev, které mě donutily přehodnotit některá rozhodnutí. Zjistil jsem například, že rozsáhlejší konfigurace pomocí Docker Compose může být časově náročná a složitá na údržbu, zejména při přidávání nových komponent clusteru. Další oblastí k vylepšení je monitoring – v budoucnu bych chtěl nasadit robustnější nástroje jako Prometheus nebo ELK stack pro lepší sledování a diagnostiku chování systému.

Projekt mi poskytl hluboký vhled do fungování distribuovaných NoSQL databází a naučil mě řešit problémy replikace, shardingu, agregací a optimalizací výkonu. Díky této práci jsem si také ověřil, že navržené řešení může být efektivně využito nejen v akademickém prostředí, ale i v reálných komerčních projektech, které vyžadují škálovatelnost, vysokou dostupnost a pokročilé analytické možnosti.

Zdroje

Kaggle: Anime Recommendations Database

<https://www.kaggle.com/CooperUnion/anime-recommendations-database>

Kaggle: Netflix Shows

<https://www.kaggle.com/shivamb/netflix-shows>

Kaggle: TMDb Movie Metadata

<https://www.kaggle.com/tmdb/tmdb-movie-metadata>

MongoDB Case Study: City of Chicago

<https://www.mongodb.com/customers/city-of-chicago>

MongoDB Case Study: Electrolux

<https://www.mongodb.com/customers/electrolux>

MongoDB Case Study: Migu Video

<https://www.mongodb.com/customers/migu-video>

Přílohy

BSQBD-MongoDB

├── BSQBD-Aronov-Oleksandr.docx

├── data

│   ├── TopAnime.csv

│   ├── TopMovies.csv

│   ├── TopNetflix.csv

│   ├── analyze\_data.py

│   ├── clean\_data.py

│   └── plots

│   ├── TopAnime.csv\_genres\_top10.png

│   ├── TopAnime.csv\_missing.png

│   ├── TopAnime.csv\_studios\_top10.png

│   ├── TopMovies.csv\_VoteAverage\_hist.png

│   ├── TopNetflix.csv\_Type\_top10.png

│   ├── TopNetflix.csv\_country\_top10.png

│   ├── TopNetflix.csv\_missing.png

│   ├── comparison\_records.png

│   └── …

├── dotazy

│   └── dotazy.md

├── funkcni\_reseni

│   ├── docker-compose.yml

│   ├── kill\_node.sh

│   ├── mongodb-build

│   │   ├── Dockerfile

│   │   └── auth

│   │   └── mongodb-keyfile

│   └── scripts

│   ├── auth.js

│   ├── bootstrap.sh

│   ├── init-configserver.js

│   ├── init-router.js

│   ├── init-shard01.js

│   ├── init-shard02.js

│   ├── init-shard03.js

│   └── load-data.sh

└──hierarhy.txt