UNIVERZITA HRADEC KRÁLOVÉ  
Fakulta informatiky a managementu

*Nasazení a inicializace MongoDB databáze*

*Bc. Martin Malíř, 1. ročník*

V Hradci Králové dne 14. 05. 2025

Obsah

[Úvod 4](#_Toc194166540)

[1 Architektura 5](#_Toc194166541)

[1.1 Schéma a popis architektury 5](#_Toc194166542)

[1.2 Specifika konfigurace 6](#_Toc194166543)

[1.2.1 CAP teorém 6](#_Toc194166544)

[1.2.2 Cluster 6](#_Toc194166545)

[1.2.3 Uzly 7](#_Toc194166546)

[1.2.4 Sharding 7](#_Toc194166547)

[1.2.5 Replikace 8](#_Toc194166548)

[1.2.6 Perzistence dat 8](#_Toc194166549)

[1.2.7 Distribuce dat 8](#_Toc194166550)

[1.2.8 Zabezpečení 8](#_Toc194166551)

[2 Funkční řešení 9](#_Toc194166552)

[2.1 Struktura 9](#_Toc194166553)

[2.1.1 Docker-compose.yml 9](#_Toc194166554)

[2.2 Instalace 12](#_Toc194166555)

[2.2.1 Předpoklady 12](#_Toc194166556)

[2.2.2 Postup instalace 12](#_Toc194166557)

[2.2.3 Známé problémy 13](#_Toc194166558)

[2.2.4 Konečné poznámky 13](#_Toc194166559)

[3 Případy užití a případové studie 14](#_Toc194166560)

[3.1 L’Oréal: Světový lídr v oblasti krásy a technologie 14](#_Toc194166561)

[3.2 Shutterfly: Přináší miliardy fotografií k životu 14](#_Toc194166562)

[3.3 Current: Dostupné a cenově výhodné finanční služby pro všechny 15](#_Toc194166563)

[4 Výhody a nevýhody 17](#_Toc194166564)

[4.1 Výhody a nevýhody vlastního řešení 17](#_Toc194166565)

[5 Další specifika 19](#_Toc194166566)

[6 Data 20](#_Toc194166567)

[6.1 Pokémon data 20](#_Toc194166568)

[7 Dotazy 23](#_Toc194166569)

[Závěr 24](#_Toc194166570)

[Zdroje 25](#_Toc194166571)

[Přílohy 26](#_Toc194166572)

Úvod

Tato práce se zabývá vytvořením a inicializací databáze MongoDB, která je realizována jako sharded cluster. V práci je popsáno řešení od základního teoretického návrhu, až po detailní konfiguraci a zabezpečení.

TODO: rozepsat více

# Architektura

Tato kapitola popisuje nasazení MongoDB databáze v rámci sharded clusteru pomocí Docker Compose.

## Schéma a popis architektury

Obsah obrázku snímek obrazovky, text, software, design

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 1 - Ilustrace architektury

MongoDB cluster v této architektuře obsahuje jeden config server, který ukládá metadata o struktuře clusteru a rozdělení dat. Tento server se pojí na replica set o třech nodách, kde jedna vystupuje v roli primárního uzlu s dvěma sekundárními uzly. Tyto uzly obsahují samotná data, která jsou rozdělena mezi jednotlivé shardy pro horizontální škálování. V poslední řade, je zde router pro obsloužení komunikace mezi klientem a zbytkem databáze. Architektura byla navržena pro vysokou dostupnost, škálovatelnost a bezpečnost s využitím keyFile autentizace.

Od doporučeného použití se implementace liší implementací tří samostatných shardů v rámci jednoho Docker Compose souboru, což usnadňuje vývoj i testování, avšak není doporučeno kvůli produkčním i provozním omezením. Například, v této konfiguraci, běží všechny komponenty na jednom serveru, který při selhání kompletně znemožní používání databáze.

## Specifika konfigurace

Tato kapitola slouží k detailnímu popsání konfigurace.

### CAP teorém

MongoDB je ve své základní konfiguraci zaměřeno na konzistenci (**C**AP – consistency). Pokud je proved zápis do databáze a následné čtení, tak za předpokladu úspěšného zápisu, vždy dostaneme na výstupu očekávaná data. To je splněno díky tomu, že MongoDB vystupuje v takzvaném režimu „single-master“, ve kterém jsou všechny operace provedené pomocí primárního uzlu přeneseny na sekundární uzly. To v praxi znamená, že pokud primární uzel selže, data nebudou propagována dál a nebude zvolen jiný primární uzel, dokud není zaručené doručení a konzistence dat.

Nicméně, MongoDB je možné nakonfigurovat tak, aby byla zajištěna dostupnost (C**A**P – availability) i odolnost proti dělení (CA**P** – partition tolerance). V tomto případě, je povoleno čtení i ze sekundárních uzlů, a to i v případě, že dojde k de-synchronizaci s primárním uzlem. Tímto způsobem lze zajistit dostupnost dat, i v případě, že primární uzel selže, avšak je potřeba mít na paměti implicitně způsobené problémy ztráty synchronizace – kde data nemusí být napříč všemi uzly stejná.

Tato konfigurace clusteru zajišťuje garanci konzistence a dostupnosti. Odolnost proti dělení je implicitně podporována pomocí replikace a shardingu. V případě selhání primárního uzlu budou data doručena pomocí jednoho ze sekundárních uzlů. Díky tomu je poskytnut konzistentní pohled na data i při výpadku některého z uzlů.

### Cluster

Cluster je skupina propojených vzájemně kooperujících databázových (nebo jiných) serverů. Tyto servery se vzájemně podílejí na zpracování a ukládání dat. Cílem clusteru je zajištění škálovatelnosti, vysoké dostupnosti a odolnosti vůči výpadkům. MongoDB používá cluster ke správě velkých objemů dat a jejich efektivnímu rozdělení mezi jednotlivé uzly. Každý cluster by se měl skládat z komponent:

* Config server
* Router (mongos)
* Shardy

Tyto komponenty společně tvoří distribuovaný systém.

V tomto řešení je použit jeden cluster, který nabízí centralizovanou správu a řízení distribuce dat. Jmenovitě pak:

* Zjednodušuje konfiguraci a administraci clusteru
* Umožňuje snadné řízení replikace a shardingu
* Snižuje režijní náklady

### Uzly

Každý cluster MongoDB se skládá z několika uzlů. V minimální konfiguraci je to:

* Config server
* Router (mongos)
* Shardy

Config server udržuje metadata o clusteru a o rozdělení dat mezi jednotlivé shardy. Router slouží ke směrování požadavků od klienta na správné shardy. Shardy jsou hlavní datové uzly, kde jsou fyzicky uložena data. Každý shard je replikovaný set zajišťující redundanci dat.

Dostatečný počet uzlů je důležitý pro zajištění škálovatelnosti a dostupnosti systému. Větší počet uzlů umožňuje distribuované zpracování požadavků, což zlepšuje výkon a zajišťuje odolnost proti výpadkům. Nicméně také zvyšuje provozní náklady a složitost systému.

V tomto řešení je využito pěti uzlů:

* 1x Config server
* 1x Router (mongos)
* 3x Shard

Tento počet uzlů zajišťuje vysokou dostupnost, redundanci a efektivní rozdělení zátěže.

### Sharding

Sharding je technika horizontálního škálování databází. Umožňuje automatické rozdělení velkého množství dat mezi shardy na základě shard klíče. To může být například unikátní identifikátor, nebo časové razítko. Tento mechanismus pomáhá zvýšit výkon dotazů, neboť každý shard zpracovává část dat, což vede k snížení zatížení jednotlivých uzlů. To implicitně zabraňuje přetížení jednoho serveru, neboť data jsou rozložena podle logického schématu.

V tomto řešení jsou použity tři shardy. To slouží k rovnoměrnému rozložení dat mezi uzly, možnosti budoucího škálování přidáním dalších shardů a redundanci a vysokou dostupnost, neboť každý shard je součástí replikačního setu.

### Replikace

Replikace je mechanismus, který zajišťuje dostupnost a odolnost vůči selhání tím, že udržuje kopie dat na více uzlech. To zajišťuje, že při selhání primárního uzlu, můžou data poskytnout ostatní, redundantní uzly.

V tomto řešení jsou tři replikační uzly. Primary, který přijímá zápisy a řídí konzistenci dat a dva secondary, které udržují aktuální kopii a mohou být využity pro čtení.

### Perzistence dat

Persistence dat znamená, že data přetrvávají uložena i po restartování systému. MongoDB pro toto používá:

* WiredTiger storage engine
* Write-Ahead Logging
* MongoDB oplog

V tomto řešení jsou data ukládána na hostitelský systém pomocí Docker Volume, což umožňuje zachování dat při restartu kontejnerů.

### Distribuce dat

Distribuce dat v MongoDB probíhá pomocí mongos routeru. Ten směruje požadavky na správné shardy na základě konfigurace v config serveru. Každý shard uchovává část dat a jejich replikované kopie pro zajištění redundance. Distribuce dat umožňuje paralelní zpracování dotazů napříč shardy a automatické rozložení mezi servery.

V tomto řešení jsou data distribuována následovně:

* Mongos router – přesměrovává dotazy na správné shardy
* Config server – udržuje metadata o rozmístění dat
* Každý shard – ukládá část dat a repliky zajištují jejich dostupnost

### Zabezpečení

Bezpečnost databázového clusteru je klíčová pro ochranu před neoprávněným přístupem. MongoDB podporuje několik způsobů, pomocí kterých je možné přístup k datům zabezpečit.

* Autentizace – keyfile, který zajišťuje, že pouze autorizované uzly mohou komunikovat v rámci clusteru.
* Role-based Access Control – omezuje přístup uživatelů na základě oprávnění
* Šifrování přenosu dat – ochrana při komunikaci mezi servery

V tomto řešení je zabezpečení implementováno pomocí keyfile – každý uzel musí mít odpovídající klíč a admin účtu s root oprávněním.

# Funkční řešení

Tato kapitola obsahuje popis návodu na zprovoznění funkčního řešení a popis jeho struktury.

## Struktura

Adresářová struktura projektu je rozdělena následovně:

* data
* dotazy
* funkcni\_reseni
  + scripts

Složka data obsahuje veškeré datové soubory, se kterými projekt pracuje. Jsou zde tři csv soubory, které obsahují data v surovém stavu před úpravou. Následně dva csv soubory se sufixem cleaned, které obsahují zpracovaná data pomocí Python skriptů. Následně jsou zde obsaženy tři Python skripty, které jsou využívány pro analýzu a vizualizaci dat a dva Python skripty se sufixem Cleaner, které slouží pro předzpracování dat před vložením do databáze.

Složka dotazy obsahuje textový soubor, ve kterém jsou obsaženy všechny dotazy s popisem v přirozeném jazyce.

Složka funkcni\_reseni obsahuje soubor docker-compose.yml, který slouží k nastartování a jednoduchému nastavení kontejnerů. Následně je zde soubor mongo-setup.sh, který slouží ke kompletní inicializaci databáze a k importu dat. Je zde také obsažena podsložka scripts. V té lze najít Dockerfile a mongo-keyfile.

### Docker-compose.yml

Tato část slouží k detailnímu popisu souboru docker-compose.yml

#### config-server

config-server:

image: mongo:latest

build:

context: ./scripts

dockerfile: Dockerfile

container\_name: config-server

command: mongod --port 27017 --configsvr --replSet config-server --bind\_ip\_all --keyFile /etc/mongo-keyfile

volumes:

- ./scripts:/scripts

- mongodb\_cluster\_config-server\_db:/data/db

- mongodb\_cluster\_config-server\_config:/data/configdb

ports:

- 27018:27017

Config server je speciální MongoDB instance, která ukládá metadata o sharded clusteru. V kontejneru config-server se používá oficiální MongoDB obraz, který je doplněn specifickými požadavky pro sestavení kontejneru. Toho je docíleno pomocí definování kontextové složky, ve které se má spustit Dockerfile, který slouží k připravení keyfile. Následně je zde definováno jméno kontejneru, pro jednodušší identifikaci a příkaz, pomocí kterého se má kontejner spouštět:

* mongod
  + Spuštění MongoDB serveru
* --port 27017
  + Použití výchozího portu
* --configsvr
  + Označuje tuto instanci jako config server
* --replSet config-server
  + Nastavuje replikační set s názvem config-server
* --bind\_ip\_all
  + Otevře přístup z jakékoliv adresy
* --keyFile /etc/mongo-keyfile
  + Zajištění použití keyfile pro autentizaci

Poté se definuje perzistentní úložiště pro data, konfiguraci a sdílený skriptový adresář obsahující Dockerfile. Ve finále se definuje přesměrování portu na hostitelském zařízení.

#### shard-primary/shard-secondary/shard-tertiary

shard-primary:

image: mongo:latest

build:

context: ./scripts

dockerfile: Dockerfile

container\_name: shard-primary

command: mongod --port 27017 --shardsvr --replSet shard-primary --bind\_ip\_all --keyFile /etc/mongo-keyfile

volumes:

- ./scripts:/scripts

- mongodb\_cluster\_shard-primary\_db:/data/db

- mongodb\_cluster\_shard-primary\_config:/data/configdb

ports:

- 27020:27017

Tato část definuje kontejner shard-primary. Kontejnery shard-secondary a shard-tertiary se oproti tomuto moc neliší, proto budou vysvětleny najednou v této kapitole.

Shard využívá oficiální obraz MongoDB a má stejné nároky pro sestavení jako popsáno v předchozí kapitole. Rozdíl nastává během volání samotného příkazu:

* --shardsvr
  + Označuje tento MongoDB server jako shard server – část sharded clusteru
* --replSet shard-primary
  + Nastavuje replikační set s názvem shard-primary

Pro kompletnost dokumentace jsou níže ukázány i ostatní shardy:

shard-secondary:

image: mongo:latest

build:

context: ./scripts

dockerfile: Dockerfile

container\_name: shard-secondary

command: mongod --port 27017 --shardsvr --replSet shard-primary --bind\_ip\_all --keyFile /etc/mongo-keyfile

volumes:

- ./scripts:/scripts

- mongodb\_cluster\_shard-secondary\_db:/data/db

- mongodb\_cluster\_shard-secondary\_config:/data/configdb

ports:

- 27021:27017

shard-tertiary:

image: mongo:latest

build:

context: ./scripts

dockerfile: Dockerfile

container\_name: shard-tertiary

command: mongod --port 27017 --shardsvr --replSet shard-primary --bind\_ip\_all --keyFile /etc/mongo-keyfile

volumes:

- ./scripts:/scripts

- mongodb\_cluster\_shard-tertiary\_db:/data/db

- mongodb\_cluster\_shard-tertiary\_config:/data/configdb

ports:

- 27022:27017

#### router

router:

image: mongo:latest

build:

context: ./scripts

dockerfile: Dockerfile

container\_name: router

command: mongos --port 27017 --configdb config-server/config-server:27017 --bind\_ip\_all --keyFile /etc/mongo-keyfile

ports:

- 27017:27017

volumes:

- ./scripts:/scripts

- mongodb\_cluster\_router\_db:/data/db

- mongodb\_cluster\_router\_config:/data/configdb

depends\_on:

- config-server

- shard-primary

- shard-secondary

- shard-tertiary

Tento kontejner je opět definován velice podobně a primárně se liší ve spouštěcím příkazu:

* --configdb config-server/config-server:27017
  + Připojuje router ke config sereru na config-server:27017

A definovanou závislostí na ostatní kontejnery. Pokud jeden z předchozích kontejnerů původně nenastartuje, není důvod startovat tento, neboť někde nejspíš došlo k chybě, nebo nečekanému chování.

#### Perzistentní úložiště

volumes:

mongodb\_cluster\_router\_db:

mongodb\_cluster\_router\_config:

mongodb\_cluster\_config-server\_db:

mongodb\_cluster\_config-server\_config:

mongodb\_cluster\_shard-primary\_db:

mongodb\_cluster\_shard-primary\_config:

mongodb\_cluster\_shard-secondary\_db:

mongodb\_cluster\_shard-secondary\_config:

mongodb\_cluster\_shard-tertiary\_db:

mongodb\_cluster\_shard-tertiary\_config:

Config server, router a shard má svoje vlastní perzistentní úložiště, ve kterém data zůstanou i po restartování kontejnerů. Využívají se adresáře /data/db pro běžná data a /data/configdb pro konfiguraci.

## Instalace

Tato kapitola slouží k popisu, jak řešení spustit.

### Předpoklady

* Libovolný systém Linux, schopen spouštět shell skripty - /bin/bash a ovládání služeb pomocí systemd
  + Starší, nebo jiné systémy používající init nelze použít
  + Alternativně WSL ve Windows
* Nainstalovaný Docker Engine
* Dostatečná uživatelská práva pro operaci s Docker Engine – alternativně přístup na uživatele root
* Nainstalovaný Python3
* Nainstalovaný GIT pro klonování repositáře
  + Alternativně možné stáhnout ZIP soubor a extrahovat soubory

### Postup instalace

* Naklonování repositáře do zvolené složky <project\_home>

cd <project\_home>  
git pull <https://github.com/MartmatiX/mongodb_malir_martin.git>

* Zapnutí služby Docker Engine

sudo systemctl start docker

* Sestoupení do adresáře funkcni\_reseni

cd <project\_home>/funkcni\_reseni

* Spuštění inicializačního skriptu

./mongo-setup.sh

### Známé problémy

Systém Windows používá jiné kódování pro bílé znaky – CR LF, než systém Linux – LF. Po naklonování z repositáře, je možné, že skript nebude možné spustit. Chybová hláška:

-bash: ./mongo-setup.sh: /bin/bash^M: bad interpreter: No such file or directory

Řešením je přepsat všechny bílé znaky z formátu CR LF na formát LF. Toho lze jednoduše docílit pomocí nástroje dos2unix

sudo apt install dos2unix

dos2unix <project\_home>/funkcni\_reseni/mongo-setup.sh

### Konečné poznámky

Shell skript má reference na několik ostatních souborů, které jsou využity během procesu inicializace. Jakákoliv změna názvů souborů, nebo adresářové struktury musí být reflektována i v Shell skriptu. Za jakékoliv vlastní úpravy, které znemožní používání skriptu zodpovídá uživatel.

# Případy užití a případové studie

MongoDB je vhodná pro různé typy aplikací díky své flexibilní struktuře a schopnosti efektivně pracovat s velkými objemy dat. Často se využívá v oblasti webových a mobilních aplikací, kde umožňuje rychlý vývoj díky absenci pevně definovaných struktur a rozsáhlé podpoře s integrací v moderních jazycích. Podpora JSON formátu také nachází uplatnění v systémech pro správu obsahu – CMS, kde umožňuje práci s různými datovými strukturami.

Další uplatnění MongoDB je v oblastech práce s velkými datovými sadami. To může být buď oblast analytiky, business inteligence, nebo i bezpečností okruhy jako SIEM apod. Databáze podporuje agregaci a indexaci, což umožňuje efektivní zpracování velkých objemů dat v reálném čase.

V mém řešení je MongoDB používána pro uložení velkých i menších datových struktur. Jiné NoSQL databáze jako Cassandra nebo Redis nebyly vybrány z důvodu absence jakýchkoliv předchozích zkušeností s nimi.

## L’Oréal: Světový lídr v oblasti krásy a technologie

V roce 2018 se L’Oréal přesunula do éry „Beauty Tech“, využívající moderní technologie a personalizaci, inkluzi a udržitelnosti v oblasti krásy. Beauty Tech zahrnuje digitální služby, rozšířené produkty, beatuy zařízení a umělou inteligenci. Bylo vytvořeno interní technické oddělení Tech Accelerator, které podporuje digitální inovace a skládá se ze dvou částí – služby a řešení.

V oblasti služeb se zaměřují na usnadnění práce zaměstnanců, například vývojem nástrojů s umělou inteligencí, nebo na optimalizaci složení produktů. Oblast služeb pak cílí na spotřebitele a prodejce. Například služba ModiFace, která umožňuje virtuální vyzkoušení různých produktů

L’Oréal ale narazil na výzvu. Aplikace potřebovala ukládat a spravovat velké množství dat v reálném čase. Původní databáze byla příliš pomalá a technologicky omezená, aby mohla zvládnout tyto požadavky, proto se firma rozhodla k migraci na MongoDB. To přineslo v průměru 40x lepší odezvu, jednodušší správu a udržitelnost a možnost rychlejší aktualizace aplikací. Migrace byla provedena postupně, aby nedošlo k narušení provozu. Byly vytvořeny odpovídající modely původní databáze a data byla přesunuta pomocí migračních skriptů.

Doba odezvy se po migraci snížila na přibližně 10 milisekund což přineslo větší uživatelskou satisfakci.

## Shutterfly: Přináší miliardy fotografií k životu

Shutterfly umožňuje milionům uživatelů ukládat, sdílet a upravovat miliardy fotografií. Nabízí širokou škálu produktů – od fotoknih, dárků a dekorací až po osobní webové stránky. S růstem objemu dat potřebovala společnost spolehlivé a škálovatelné řešení pro jejich správu.

Relační databáze se ukázaly jako nepružné a drahé na provoz. Shutterfly proto přešel na dokumentovou databázi MongoDB, která nabízí flexibilitu a možnost horizontálního škálování. MongoDB umožňuje ukládat data intuitivně a přizpůsobovat databázi potřebám aplikace, což usnadňuje vývoj.

Společnost zvolila MongoDB Atlas na AWS, kde provozuje několik clusterů až o velikosti 6 TB. Pomocí nástroje Mongomirror byla data přenesena během několika minut bez výpadků služeb. Nová platforma nyní zvládá tisíce transakcí za minutu a přizpůsobuje se sezónním výkyvům poptávky, například během Vánoc.

Přechod na MongoDB Atlas přinesl 20% snížení provozních nákladů, vyšší flexibilitu a rychlejší zavádění nových služeb. Společnost nyní snadno škáluje kapacitu během špiček a plánuje rozšíření do multi-cloud prostředí, což umožní poskytovat služby různým partnerům.

## Current: Dostupné a cenově výhodné finanční služby pro všechny

Finančně-technologická společnost Current vznikla s cílem zpřístupnit bankovní služby široké veřejnosti. Spolupracuje s menšími bankami po celých Spojených státech a propojuje zákazníky s moderními technologiemi, které usnadňují správu financí. Platforma Current je navržena tak, aby vyhovovala potřebám čtyř milionů uživatelů, přičemž klade důraz na dostupnost, flexibilitu a nízké náklady.

Tradiční banky jsou zaměřené na účty, nikoli na zákazníky. Kvůli neustálým změnám v personálu musí klienti opakovaně sdělovat své informace novým poradcům, což vede ke zmatkům a komplikacím. Tento problém ještě prohlubují zastaralé IT systémy, které vytvářejí datová a organizační omezení. Current se rozhodl tento model změnit – chtěl vytvořit digitální bankovnictví budoucnosti, které zákazníkům poskytne jednoduché a efektivní služby bez zbytečných překážek. K tomu ale potřeboval moderní technologickou infrastrukturu.

Current vyvinul vlastní bankovní jádro Current Core, které spravuje účetní záznamy a transakce. K jeho provozu potřeboval rychlou a škálovatelnou databázi, a proto zvolil MongoDB Atlas. Mezi klíčové výhody MongoDB Atlas patří:

* Flexibilní model dokumentů, který zvládá různé typy transakcí – od karetních plateb po kryptoměnové operace.
* ACID transakce zajišťující integritu dat.
* Atlas Search pro rychlé a přesné dotazy.
* MongoDB Compass pro snadnou analýzu dat.

Celý systém je postaven na událostně řízené architektuře, která zaznamenává každou transakci do MongoDB Atlas. Díky tomu lze efektivně spravovat bankovní operace, včetně ACH převodů, mobilních šekových vkladů, peer-to-peer plateb a výběrů z bankomatů.

Current provozuje svůj systém na Google Cloud, kde využívá Google Kubernetes Engine a další cloudové služby jako Dataflow, PubSub a BigQuery. MongoDB Atlas umožňuje automatické škálování bez nutnosti zásahu provozního týmu, čímž snižuje provozní náklady.

# Výhody a nevýhody

Jednou z hlavních předností MongoDB je flexibilní schéma, které umožňuje ukládání dokumentů s různou strukturou bez nutnosti složitých migrací. To je obzvláště výhodné pro aplikace, které se neustále vyvíjejí a potřebují dynamicky měnit datový model. Další významnou výhodou je vysoká škálovatelnost – MongoDB podporuje horizontální škálování pomocí shardingu, což umožňuje efektivní distribuci dat mezi více servery a zajišťuje stabilní výkon i při práci s velkým objemem dat. Díky indexaci a binárnímu formátu BSON je databáze optimalizovaná pro rychlé operace čtení a zápisu, což ji činí vhodnou pro aplikace vyžadující vysokou propustnost, například v oblasti analytiky či zpracování velkých dat. Kromě toho MongoDB podporuje replikaci pomocí tzv. replica setů, což zvyšuje dostupnost a odolnost proti selhání serveru.

Nicméně, MongoDB přináší i určité nevýhody, které je nutné zvážit při jejím nasazení. Jedním z potenciálních problémů je vyšší spotřeba paměti, jelikož BSON formát a absence tradičních relačních omezení mohou vést k větší velikosti uložených dat. Další nevýhodou je omezená podpora transakcí – ačkoliv novější verze MongoDB již podporují ACID transakce, stále nejsou tak robustní jako u tradičních relačních databází, což může být nevýhodné pro aplikace vyžadující silnou konzistenci dat. Také samotný dotazovací jazyk MongoDB se liší od SQL, což může být pro vývojáře zvyklé na relační databáze výzvou. Další potenciální problém spočívá v modelu eventual consistency, který znamená, že změny v databázi nemusí být okamžitě viditelné ve všech kopiích, což může být nevhodné pro aplikace s vysokými nároky na konzistenci, jako jsou například bankovní systémy

## Výhody a nevýhody vlastního řešení

Jednou z hlavních výhod tohoto řešení je modularita a škálovatelnost. Zvolený clusterový přístup umožňuje snadné přidávání dalších shardů bez zásadní změny architektury, což podporuje horizontální škálování. Použití shardingu umožňuje distribuovat data mezi více databázovými uzly, což vede k lepšímu výkonu při práci s velkým množstvím dat a vyšší dostupnosti systému.

Další výhodou je zabezpečení a izolace prostředí. Konfigurace využívá keyFile, což znamená, že jednotlivé instance MongoDB komunikují šifrovaně a ověřují svou identitu, což zvyšuje bezpečnost nasazení. Kromě toho jsou data a konfigurace uchovávány ve vyhrazených volume, což umožňuje perzistenci dat i po restartu kontejnerů.

Výhodou je také jednoduchost nasazení. Díky Docker Compose lze celé řešení spustit jedním příkazem, což snižuje složitost ruční konfigurace sharded clusteru.

Jedním z problémů tohoto řešení je vyšší složitost správy. I když Docker Compose usnadňuje nasazení, konfigurace sharded clusteru MongoDB vyžaduje další kroky, jako je inicializace replikačních setů a shardování kolekcí pomocí manuálních příkazů v Mongo shellu. Bez této inicializace cluster nebude správně fungovat.

Další nevýhodou je vyšší spotřeba zdrojů. Každý shard běží jako samostatný kontejner, což znamená, že při provozu více shardů na jednom fyzickém nebo virtuálním stroji může dojít k vysoké spotřebě CPU a RAM. Toto řešení je proto vhodnější pro nasazení v distribuovaném prostředí s více servery než na jediném vývojářském stroji.

# Další specifika

Toto zvolené řešení je navržené pro stabilní testování, nebo jako základ pro vetší škálovatelné nasazení MongoDB. Avšak pro produkční nasazení by bylo potřeba srovnat některá rozhodnutí s těmi, které jsou uvedeny v oficiálních zdrojích. Jako příklad je to „single point of failure“ config server nebo router, které nemají svého náhradníka, a tudíž jejich selhání znamená výpadek komunikace mezi databází a klienty.

# Data

Tato kapitola slouží pro popis a analýzu použitých datových sad.

## Pokémon data

Název souboru: pokemon.csv

Cesta k souboru: <project\_home>/data/pokemon.csv

Název kolekce: Pokemon

Datové typy:

|  |  |
| --- | --- |
| # | int |
| Name | string |
| Type 1 | string |
| Type 2 | string, null |
| Total | int |
| HP | int |
| Attack | int |
| Defense | int |
| Sp. Atk | int |
| Sp. Def | int |
| Speed | int |
| Generation | int |
| Legendary | string |

Počet záznamů: 721

Prázdné hodnoty:

|  |  |
| --- | --- |
| # | 0 |
| Name | 0 |
| Type 1 | 0 |
| Type 2 | 386 |
| Total | 0 |
| HP | 0 |
| Attack | 0 |
| Defense | 0 |
| Sp. Atk | 0 |
| Sp. Def | 0 |
| Speed | 0 |
| Generation | 0 |
| Legendary | 0 |

Úpravy: žádné

Zdroj: <https://gist.github.com/armgilles/194bcff35001e7eb53a2a8b441e8b2c6>

Ukázka dat:



Obrázek 2 - Ukázka dat – Pokémon

Rozdělení celkového skóre mezi Pokémony:

Obsah obrázku diagram, Vykreslený graf, snímek obrazovky, řada/pruh

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 3 - Distribuce skóre Pokémonů

Počty Pokémonů podle primárního typu:

Obsah obrázku text, snímek obrazovky, řada/pruh, diagram

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 4 - Počet Pokémonů podle typu

Korelační matice statistik Pokémonů:

Obsah obrázku text, snímek obrazovky, čtverec, Obdélník

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 5 - Korelační maticke Pokémonů

Průměrné statistiky Pokémonů podle generace:

Obsah obrázku text, snímek obrazovky, Barevnost, Vykreslený graf

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 6 - Průměrné statistiky generací

## Video Games Sales

Název souboru: vgsales.csv

Cesta k souboru: <project\_home>/data/vgsales.csv

Název očištěného souboru: vgsales\_cleaned.csv

Cesta k očištěnému souboru: <project\_home>/data/vgsales\_cleaned.csv

Název kolekce: games

Datové typy:

|  |  |
| --- | --- |
| Rank | int |
| Name | string |
| Platform | string |
| Year | double |
| Genre | string |
| Publisher | string |
| NA\_Sales | double |
| EU\_Sales | double |
| JP\_Sales | double |
| Other\_Sales | double |
| Global\_Sales | double |

Počet záznamů: 16598

Prázdné hodnoty:

|  |  |
| --- | --- |
| Rank | 0 |
| Name | 0 |
| Platform | 0 |
| Year | 271 |
| Genre | 0 |
| Publisher | 58 |
| NA\_Sales | 0 |
| EU\_Sales | 0 |
| JP\_Sales | 0 |
| Other\_Sales | 0 |
| Global\_Sales | 0 |

Úpravy:

Ve své původní podobě měl dataset chybějící hodnoty nastavené na „N/A“ místo prázdné hodnoty null. Z toho důvodu nebylo možné 271 dokumentů vložit do databáze, neboť požadovaná hodnota pro Year je int. N/A je však typu string. Byl použit jednoduchý skript pro nahrazení této hodnoty za null, poté proběhlo vložení dokumentů v pořádku. Skript také upravil hodnotu Year z formátu YYYY.0 na YYYY a všechny položky přetypoval na požadovanou hodnotu pro zajištění kompatibility s nastaveným validátorem.

import pandas as pd  
  
  
def clean\_csv(input\_file, output\_file):  
 df = pd.read\_csv(input\_file)  
  
 df.replace("N/A", None, inplace=True)  
 df["Rank"] = pd.to\_numeric(df["Rank"], errors="coerce").astype("Int64")  
 sales\_columns = ["NA\_Sales", "EU\_Sales", "JP\_Sales", "Other\_Sales", "Global\_Sales"]  
 for col in sales\_columns:  
 df[col] = pd.to\_numeric(df[col], errors="coerce").astype(float)  
 df["Year"] = pd.to\_numeric(df["Year"], errors="coerce").astype("Int64")  
  
 df.to\_csv(output\_file, index=False)  
 print(f"Cleaned CSV saved as {output\_file}")  
  
  
if \_\_name\_\_ == "\_\_main\_\_":  
 input\_file = "vgsales.csv"  
 output\_file = "vgsales\_cleaned.csv"  
 clean\_csv(input\_file, output\_file)

Po dokončení skriptu je vytvořen další soubor vgsales\_cleaned.csv, který se importuje do samotné databáze.

Zdroj: <https://www.kaggle.com/datasets/anandshaw2001/video-game-sales>

Ukázka dat:



Obrázek 7 - Ukázka dat - prodej videoher

Počet her podle platformy:

Obsah obrázku text, snímek obrazovky, Vykreslený graf, řada/pruh

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 8 - Počet her podle platformy

Korelační matice prodeje:

Obsah obrázku text, snímek obrazovky, čtverec, Obdélník

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 9 - Korelační matice prodejů videoher

Průměrná celosvětová prodejnost:

Obsah obrázku text, snímek obrazovky, číslo, Písmo

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Obrázek 10 - Průměrná prodejnost videoher

## Zdravotní data

# Dotazy

Xxxxxx

Závěr

Xxxxxx

Zdroje

Xxxxxx

Přílohy

[Obrázek 1 - Ilustrace architektury 5](#_Toc194241504)

[Obrázek 2 - Ukázka dat – Pokémon 21](#_Toc194241505)

[Obrázek 3 - Distribuce skóre Pokémonů 21](#_Toc194241506)

[Obrázek 4 - Počet Pokémonů podle typu 22](#_Toc194241507)

[Obrázek 5 - Korelační maticke Pokémonů 22](#_Toc194241508)

[Obrázek 6 - Průměrné statistiky generací 23](#_Toc194241509)

[Obrázek 7 - Ukázka dat - prodej videoher 25](#_Toc194241510)

[Obrázek 8 - Počet her podle platformy 25](#_Toc194241511)

[Obrázek 9 - Korelační matice prodejů videoher 25](#_Toc194241512)

[Obrázek 10 - Průměrná prodejnost videoher 26](#_Toc194241513)