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 3](#_Toc193994772)

[1 Architektura 4](#_Toc193994773)

[1.1 Schéma a popis architektury 4](#_Toc193994774)

[1.2 Specifika konfigurace 5](#_Toc193994775)

[1.2.1 CAP teorém 5](#_Toc193994776)

[1.2.2 Cluster 5](#_Toc193994777)

[1.2.3 Uzly 6](#_Toc193994778)

[1.2.4 Sharding 6](#_Toc193994779)

[1.2.5 Replikace 7](#_Toc193994780)

[1.2.6 Perzistence dat 7](#_Toc193994781)

[1.2.7 Distribuce dat 7](#_Toc193994782)

[1.2.8 Zabezpečení 7](#_Toc193994783)

[2 Funkční řešení 8](#_Toc193994784)

[2.1 Struktura 8](#_Toc193994785)

[2.1.1 Docker-compose.yml 8](#_Toc193994786)

[2.2 Instalace 11](#_Toc193994787)

[2.2.1 Předpoklady 11](#_Toc193994788)

[2.2.2 Postup instalace 11](#_Toc193994789)

[2.2.3 Známé problémy 12](#_Toc193994790)

[2.2.4 Konečné poznámky 12](#_Toc193994791)

[3 Případy užití a případové studie 13](#_Toc193994792)

[4 Výhody a nevýhody 14](#_Toc193994793)

[5 Další specifika 15](#_Toc193994794)

[6 Data 16](#_Toc193994795)

[7 Dotazy 17](#_Toc193994796)

[Závěr 18](#_Toc193994797)

[Zdroje 19](#_Toc193994798)

[Přílohy 20](#_Toc193994799)

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.

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

TODO: dopsat tuhle část, teď nevím, co tu má být

### 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

# Výhody a nevýhody

Xxxxxx

# Další specifika

Xxxxxx

# 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

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:



Rozdělení celkového skóre mezi Pokémony

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

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Počty Pokémonů podle primárního typu:

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

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Korelační matice statistik Pokémonů:

Obsah obrázku text, snímek obrazovky, čtverec, Obdélník

Obsah vygenerovaný umělou inteligencí může být nesprávný.

Průměrné statistiky Pokémonů podle generace:

Obsah obrázku text, snímek obrazovky, Barevnost, Vykreslený graf

Obsah vygenerovaný umělou inteligencí může být nesprávný.

# Dotazy

Xxxxxx

Závěr

Xxxxxx

Zdroje

Xxxxxx

Přílohy

Xxxxxx