Fakulta informačních technologií
Vysoké učení technické v Brně

Pokročilé databázové systémy – Projekt
Návrh aplikace

1. prosince 2024

Tomáš Vojík
xvojik00

1 Specifikace aplikace

Cílem je vytvořit aplikaci sloužící pro optimalizaci distribuce a výroby ve výrobních řetězcích.

Systém se skládá z několika propojených výrobních míst (továren), které produkují nějaké výstupy a mohou vyžadovat i vstupy. Speciálním typem továrny je finální distribuční centrum, které nemá žádné výstupy, ale pouze spotřebovává své vstupy (např. prodej koncovým zákazníkům → generuje zisk). Každá továrna vytváří přesně jeden typ produktu s konstantní rychlostí, pokud má dostatek vstupů. Továrny mají omezené skladové kapacity pro vstupy a výstupy. Továrny jsou propojeny distribuční sítí, která se skládá z několika různých spojů, které mají různou kapacitu a rychlost (např. vlaky, lodě, kamiony). Spoje jsou pouze jednosměrné. Jeden spoj může v jednu chvíli obsluhovat jednu přepravu, ale může převážet i několik typů materiálu do své maximální kapacity. Továrny mohou sloužit i jako překladová místa pro více navazujících spojů.

Aplikace by měla poskytovat přehled o distribuční síti za účelem optimalizace přepravy materiálů mezi továrnami s cílem vytvoření cílového produktu pro distribuční centra a generování zisku.

1.1 Seznam operací nad daty

Níže definujeme potřebné dotazy a operace, které musí aplikace umět provádět.

1.1.1 Seznam dotazů

1. **Seznam všech továren**: S možností filtrovat na základě požadovaných vstupů, výstupů a názvu, ap.
   - *Typické použití*: Získání přehledu o továrnách. Vyhledání továren, které vyžadují určitý vstup / poskytují určitý výstup.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že dotaz vrátí všechny továrny bez použití filtru.
     - Ověřit funkčnost filtru podle jednotlivých filtrovatelných kategorií.
     - Testování více filtrů zároveň (např. továrny, které vyžadují vstupy A a B a poskytují výstup C).
     - Ověřit, že při použití neexistujícího vstupu/výstupu jako filtru dotaz vrátí prázdnou množinu.
     - Po přidání nové továrny ověřit, že se objeví ve výpisu.
2. **Seznam zastavených továren**:
   - *Typické použití*: Získání informací o továrnách, které zrovna nemohou nic vyrábět, protože čekají na materiály, nebo mají plný sklad.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že dotaz vrátí všechny továrny zastavené továrny.
     - Ověřit, že při doručení chybějících materiálů se továrna spustí.
     - Ověřit, že při uvolnění plného skladu se továrna spustí.
3. **Výpis typů materiálů**: Výpis typů materiálů, které se používají ve výrobním procesu.
   - *Typické použití*: Přehled o materiálech, které továrny zpracovávají a jejich vlastnostech.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že dotaz vrátí kompletní seznam materiálů uložených v systému.
     - Po přidání nového materiálu ověřit, že se objeví ve výpisu.
     - Ověřit správnost zobrazených vlastností materiálů.
4. **Aktuální stav skladu továrny**: Vypíše aktuální skladové zásoby a volné místo pro konkrétní továrnu.

- *Typické použití*: Kontrola skladových zásob továrny k plánování dodávek (dovoz nových materiálů / vývoz vyrobených).
- *Testy pro ověření funkčnosti*:
  - Ověřit, že dotaz vrátí správné množství skladových zásob pro danou továrnu.
  - Simulovat naskladnění a vyskladnění a ověřit, že se změny promítnou do stavu skladu.
  - Ověřit, že systém správně počítá volnou kapacitu skladu.

5. **Výpis možných spojů mezi továrnami**: Vypíše všechny dostupné spoje mezi továrnami. Možnost vyhledání spojení pro jednu továrnu nebo mezi dvěma továrnami. Možnost filtrování již využitých spojů.

   - *Typické použití*: Získání informací o možných spojích pro plánování dopravy.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že dotaz vrátí všechny dostupné spoje bez použití filtrů.
     - Ověřit funkčnost filtru pro výběr spojů od/do konkrétní továrny.
     - Ověřit, že filtrování již využitých spojů správně vyloučí obsazené spoje.
     - Testovat vyhledávání spojů mezi dvěma konkrétními továrnami, které na sebe přímo navazují.
     - Testovat vyhledávání spojů mezi dvěma konkrétními továrnami, které na sebe přímo nenavazují (cesta vede přes další továrny).

1.1.2 Seznam operací

1. **Přidání/úprava továrny**

   - *Typické použití*: Evidence nové továrny, nebo úprava výrobního procesu továrny např. při zavedení nového efektivnějšího procesu, navýšení kapacity výroby,...
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že nová továrna je správně uložena v databázi a objeví se ve výpisu továren.
     - Po úpravě továrny ověřit, že změny se promítly do systému a jsou viditelné v souvisejících dotazech.
     - Testovat, zda systém správně validuje vstupní hodnoty.
     - Ověřit, že úpravy továrny nevedou k inkonzistencím v datech (např. skladové zásoby odpovídají novým kapacitám).

2. **Přidání/úprava typu materiálu**

   - *Typické použití*: Zavedení nového produktu nebo materiálu k výrobě. Úprava velikosti materiálu.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že nový materiál je správně uložen a dostupný v systému.
     - Ověřit, že materiál lze přiřadit k továrnám a výrobním procesům.
     - Testovat, zda systém správně validuje vstupní hodnoty.
     - Po úpravě materiálu ověřit, že změny jsou reflektovány v továrnách, které materiál používají.
     - Testovat, jak systém reaguje na změnu kritických parametrů materiálu (např. změna hmotnosti ovlivní kapacitu dopravy).
     - Ověřit, že úprava materiálu neovlivní historická data nesprávným způsobem.

3. **Přidání/úprava dopravního spoje mezi továrnami**: Pro spoje s aktivní přepravou by mělo být možné měnit pouze parametry spoje, nikoliv továrny, které spoj spojuje.

   - *Typické použití*: Zavedení nového spoje. Úprava aktuálního (např. zrychlení, zvýšení kapacity,...).
   - *Testy pro ověření funkčnosti*:

  - Ověřit, že nový spoj je správně uložen a zobrazuje se ve výpisu možných spojů.
  - Po úpravě spoje ověřit, že změny se promítly do plánování přeprav a že nedochází k překročení nových omezení.
  - Testovat, zda systém správně validuje vstupní hodnoty.
  - Testovat, že systém nedovolí upravit výchozí body spoje s aktivní přepravou.

4. **Přiřazení/odebrání spoje k přepravě materiálu**

   - *Typické použití*: Nastavení spoje, který má obsluhovat přepravu z jedné továrny do další a nastavení maximálního množství jednotlivých materiálů k naložení.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že přiřazení spoje k přepravě je správně uloženo a spoj je označen jako obsazený.
     - Ověřit, že naložené množství materiálů nepřekračuje kapacitu spoje ani skladové zásoby.
     - Ověřit, že nelze přiřadit jeden spoj vícekrát.
     - Testovat přepravu více typů materiálů současně a ověřit správné rozdělení kapacity.
     - Po odebrání spoje ověřit, že je uvolněn pro další přepravy a že stav materiálů je aktualizován (např. v případě zrušení přepravy se materiály vrátí do skladu).

5. **Spuštění/ukončení přepravy**: Umožňuje nastavit naložený materiál. V případě vykládky vždy vykládá veškerý materiál pokud je na něj místo na skladě.

   - *Typické použití*:
     - Označení spoje, který vyjel z jedné továrny směrem do druhé.
     - Označení spoje, který ukončil svou jízdu.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že spoj se označí jako aktivní, ponížil stav skladu výchozí továrny, má nastavené správné množství převáženého materiálu a nepřesáhl svou maximální kapacitu.
     - Ověřit, že spoj vyloží veškerý svůj náklad v cílové továrně, změní stav skladu cílové továrny a označí se jako neaktivní.
     - Ověřit, že pokud továrna nemá dostatečnou skladovou kapacitu, spoj vyloží jen tolik materiálu, kolik může, změní stav skladu cílové továrny a zůstává aktivní.

6. **Přidání/odebrání skladových zásob továrny**:

   - *Typické použití*:
     - Továrna vyrobila určité množství výrobku a přesunula ho na sklad.
     - Distribuční centrum prodalo určité množství výrobku.
   - *Testy pro ověření funkčnosti*:
     - Ověřit, že se korektně upraví stav skladu továrny.
     - Ověřit, že továrna nedovolí naskladnit více, než je její maximální kapacita skladu.
     - Ověřit, že nelze naskladněný materiál ponížit o větší množství, než je naskladněno.

1.2 Testování aplikace

Pro zajištění kvality a spolehlivosti aplikace je nutné provést důkladné testování na různých úrovních. Testování se zaměřuje na jednotlivé komponenty systému, jejich integraci a na systém jako celek. Následující podsekce popisují strategie testování, které budou použity:

1. **Jednotkové testy (Unit Testing)**: Jednotkové testy ověřují správnou funkčnost jednotlivých modulů nebo funkcí aplikace v izolaci. Cílem je zajistit, že každý modul funguje správně samostatně.

2. **Integrační testy (Integration Testing)**: Integrační testy ověřují správnou spolupráci mezi jednotlivými komponentami systému.

3. **Funkční testy (Functional Testing)**: Funkční testy jsou zaměřeny na ověření, že systém splňuje požadované funkční požadavky a správně zpracovává definované operace a dotazy.

4. **Výkonnostní testy (Performance Testing)**: Výkonnostní testy ověřují schopnost systému zvládat očekávanou zátěž a identifikují možné výkonnostní problémy.

5. **Testy spolehlivosti a dostupnosti (Reliability and Availability Testing)**: Tyto testy ověřují odolnost systému vůči výpadkům a schopnost zotavení po chybách.

6. **Testování celého systému (End-to-End Testing)**: End-to-End testy ověřují funkčnost celého systému z pohledu uživatele nebo externího systému.

2 Návrh architektury

Navržená architektura musí dodržovat principy `CQRS` a využívá `Event Sourcing`. V relační databázi využívá pro ukládání většiny dat aplikace. Pro ukládání distribuční sítě využijeme grafovou `NoSQL` databázi.

Na obrázku 1 je znázorněna základní architektura celé aplikace. Architektura je rozdělena na dvě základní části dle `CQRS`: `Command side` a `Query Side`. Přidáná je také `Simulator Side`, která zajišťuje průběh simulace (viz 2.1).

`Command Side` obsahuje `Message Queue`, která zajišťuje postupné zpracování požadavků pomocí procesoru (`Event Processor`), které mohou být i distribuované. Dle `Event Sourcing` je každá událost zaznamenávána do `Event Store`.

`Query Side` obsahuje `Data Model`, který má na starost zpracování dat z databází do cílové podoby. Následně data procházejí přes `Cache` vrstvu.

2.1 Simulace

Pro jednodušší práci a testování systému obsahuje aplikace i simulační část. Tato část by v reálném prostředí byla nadbytečná.

Simulátor by měl být manuálně nebo v nějakém časovém intervalu být schopný spouštět jeden krok simulace. U továren se předpokládá, že za jeden krok simulace dokážou dokončit přesně jeden výrobní proces (zpracovat materiály a vytvořit produkt). Spoje mají uvedenou svou rychlost (počet kroků simulace, než dokončí dodávku). Dle rychlosti a dat z `Event Store` může simulátor vyhodnocovat, které akce se mohou automaticky provést.

Např. pokud má továrna naskladněné všechny potřebné materiály a volné místo na skladě, může provést výrobní proces: odebere materiály ze skladu a přidá produkt. Přiřazený neaktivní spoj může v dalším kroku ze skladu odebrat produkt a zahájit cestu do cílové továrny. Aktivní spoje, které převáží materiál a vyrazili před x kroky simulace, kde $x \geq$ rychlost spoje, se mohou vyložit v cílové továrně.

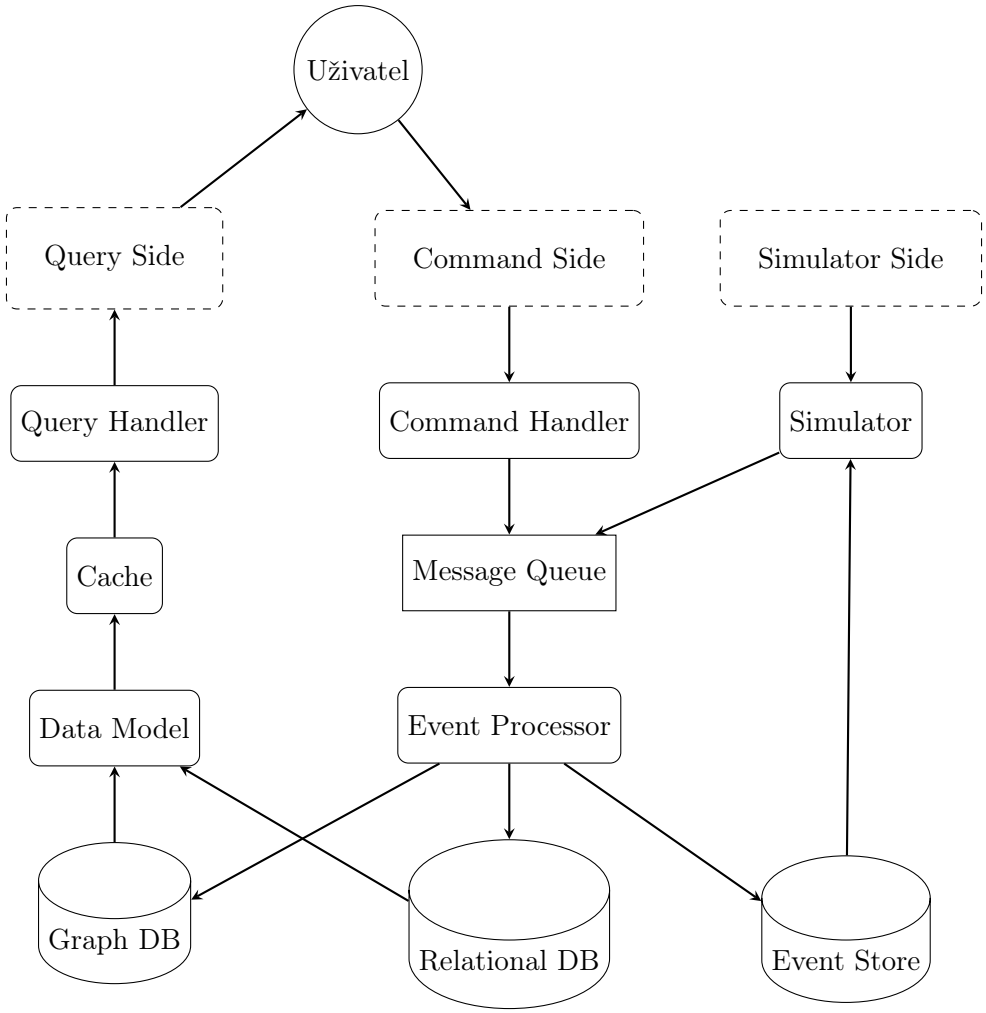
Simulátor bude navíc při každém svém kroku do `Event Store` ukládat speciální událost, která symbolizuje počítadlo kroků simulace.

2.2 Rizika

Navržená architektura systému přináší několik potenciálních rizik, která mohou ovlivnit funkčnost, výkon nebo spolehlivost aplikace. Níže jsou popsaná některá z rizik, kterým může aplikace čelit a jejich mitigace vzhledem k navržené architektuře.

2.2.1 Riziko datové nekonzistence mezi relační a nerelační databází

Popis rizika: Vzhledem k tomu, že data jsou uložena v různých databázových systémech (relační, event store, grafová a cache), může dojít k situacím, kdy data nejsou synchronizována nebo jsou v nekonzistentním stavu.



Obrázek 1: Základní struktura architektury

Mitigace:

- **Implementace Event Sourcingu**: Všechny změny jsou zaznamenávány jako události v `Event Store`, což umožňuje rekonstruovat stav systému a zajistit konzistenci.
- **Asynchronní zpracování událostí**: Použití `Event Processorů` k zajištění sekvenčního a idempotentního zpracování událostí.

2.2.2 Nedostatečný výkon

Popis rizika: Některé komponenty systému mohou omezovat celkový výkon (např. relační databáze při vysokém zatížení nebo message queue při velkém počtu zpráv).

Mitigace:

- **Horizontální škálování**: Nasazení více instancí služeb a databází pro rozložení zátěže.
- **Optimalizace dotazů**: Využití indexů a optimalizace databázových dotazů pro zvýšení rychlosti přístupu k datům.
- **Caching**: Použití `cache` pro ukládání často používaných dat a snížení zátěže na databáze.
- **Asynchronní zpracování**: Využití asynchronních operací pro nezablokování hlavních procesů při čekání na dokončení náročných úloh.

2.2.3 Jednotlivé body selhání (Single Points of Failure)

Popis rizika: Pokud některé komponenty nejsou replikovány nebo nemají zálohu, může jejich výpadek způsobit nedostupnost celé aplikace.

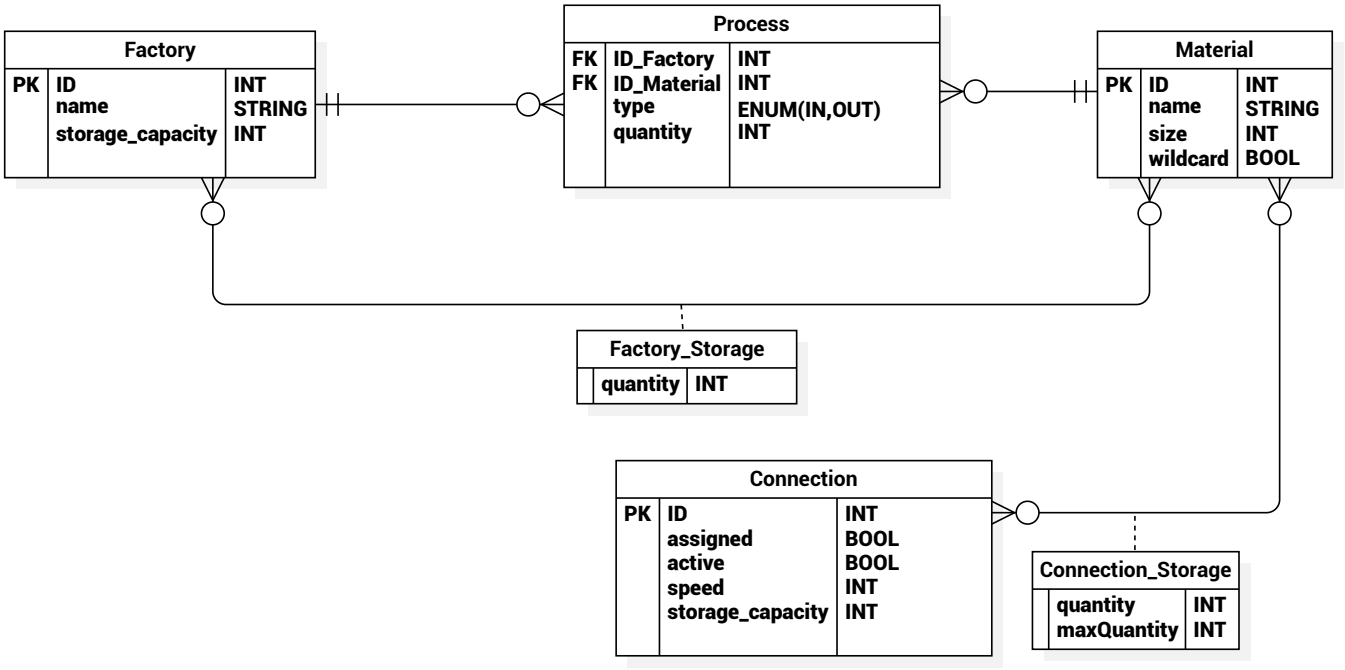
Mitigace:

- **Redundance komponent**: Nasazení clusterů pro databáze (relační, grafová a event store) a služeb.
- **Load balancing**: Použití webového serveru pro rozložení zátěže mezi aplikační servery a zajištění dostupnosti služeb.

3 Návrh databáze

3.1 Relační data

Na obrázku 2 můžeme vidět ER diagram, který reprezentuje schéma relační struktury databáze. Schéma je velmi jednoduché. Nachází se zde jen tři hlavní entity: `Factory`, `Material`, `Connection` a entita pro uchovávání informací o výrobních procesech továren: `Process`. Databáze bude navíc obsahovat dvě tabulky pro `n:m` relace, které znázorňují aktuální stav skladu továren a spojů.



Obrázek 2: ER Diagram relační databáze

Od relační databáze se očekává hlavně vysoká konzistence dat a zápisu. K tomu se bude využívat transakcí. Čtení z databáze bude bude skryté za distribuovanou vrstvou `Cache`, která bude vysokou zajišťovat dostupnost. Není proto nezbytně nutné samotnou relační databázi škálovat.

3.2 Grafová data

Grafová data aplikace obsahují informace o distribuční síti. Jednotlivé uzly tedy představují továrny (`Factory`) a hrany představují spojení (`Connection`). Spojení továren bychom mohli reprezentovat i v rámci relační databáze, ale za účelem snížení duplicitních dat a zefektivnění prohledávání grafu bude optimálnější použít samostatnou grafovou databázi.

Pro identifikaci dat musíme uzlům i hranám přidat atributy `ID`, které budou odpovídat relačním entitám. Výhodné by také mohlo být kopírovat atributy pro hrany (entita `Connection`), zejména `speed` a `storage_capacity`, jelikož ty lze použít jako váhu při vyhledávání nejoptimálnější cesty.

V grafové databázi budeme typicky jen vyhledávat možné cesty mezi jednotlivými továrnami. Dá se předpokládat vyšší množství dotazů spíše než zápisů do databáze. U grafové databáze je stejně jako u relační menší potřeba škálování, jelikož budou dotazy na ní skryté pod `cache` vrstvou.

3.3 Event Store

`Event Store` slouží jako zdroj pravdy pro celou aplikaci. Pro `Event Store` lze využít sloupcovou, nebo jinou specializovanou databázi. Každá událost bude mít následující strukturu:

1. **ID**: Identifikátor události.
2. **type**: Název nebo kategorie události (např. `FactoryCreated`, `SimulationStep`,...).
3. **data**: `JSON`, nebo jiný strukturovaný formát obsahující detailní data události.
4. **timestamp**: Časové razítko (v případě simulace se jedná o pořadové číslo kroku simulace).

Události budou vytvářené uživatelským požadavky (případně simulátorem). Jeden požadavek může vytvořit i několik událostí. Od `Event Store` se tedy očekává vysoká dostupnost pro zápis dat spíše než čtení.

3.4 Cache

Cílem `Cache` je distribuovaně a rychle poskytovat přístup k dotazovaným datům. Může se jednat o jednoduchou `NoSQL` databázi typu `Key-Value`, kde klíčem by byl nějaký identifikátor dotazu, nebo ID objektu a hodnota by obsahovala serializovaná data.

V případě cache je třeba řešit i její invalidací. Toho můžeme docílit např. nastavením `TTL`, nebo manuální invalidací při změně dat. Aplikace může reagovat na události, které zpracovává a automaticky invalidovat cache, které se těchto dotazů týkají. Lze využít žurnál, který bude obsahovat jednotlivé klíče, související s určitými značkami: např. továrna s ID 1. Při změně dat můžeme v žurnálu snadno vyhledat všechny klíče, které souvisí s danou značkou. Tento žurnál může být uložený přímo v `cache` (preferovaný způsob), nebo externě.

Aplikace bude využívat `Cache-first` přístup, kdy vždy nejprve dotáže data z `cache` a až v případě, kdy data neexistují se dotazuje datový model. Některé časté dotazy lze aktualizovat v `cache` i předběžně, pokud se změnili v databázi. Primární vlastností bude vysoká dostupnost za cenu aktuálnosti získaných dat.

4 Technologie

Aplikace bude vytvořena jako webová s `REST API` rozhraním a nebude nabízet žádné UI. Aplikace počítá s možností horizontálního škálování svých komponent pro zajištění vysoké dostupnosti a výkonu.

Technologie jsou vybírány především na základě předchozích zkušeností. Celá aplikace může být nasazena snadno pomocí Docker kontejnerů.

4.1 Backend řešení

Pro backend řešení bude použito PHP 8.3[1] s využitím serveru RoadRunner[2]. RoadRunner je výkonný aplikační server napsaný v jazyce Go.

4.1.1 Výhody použití PHP 8.3 a RoadRunneru:

- **Efektivita vývoje**: Ve vývoji v PHP mám nejvíce zkušeností a Roadrunner už jsem používal i na jiné projekty.

[1] https://www.php.net/
[2] https://roadrunner.dev/

- **Vysoký výkon**: RoadRunner poskytuje vyšší výkon díky persistentním workerům a asynchronnímu zpracování, což snižuje režii spojenou s opakovaným spouštěním PHP skriptů.
- **Asynchronní zpracování**: Podpora paralelního zpracování úloh umožňuje efektivní využití systémových prostředků.
- **Snadná integrace**: Kompatibilita s existujícími PHP frameworky a knihovnami usnadňuje vývoj a údržbu aplikace. Zároveň Roadrunner podporuje přímé napojení na další komponenty aplikace (např. `Message Queue` nebo `Cache`).

Aplikace může být nasazena za NGINX[3] reverse proxy, která poskytne další vrstvu pro zpracování HTTP požadavků, load balancing a zabezpečení.

Výhody použití NGINX reverse proxy:

- **Load balancing**: Umožňuje rozložení zátěže mezi více backend serverů, čímž zvyšuje škálovatelnost aplikace.
- **Zabezpečení**: Poskytuje možnost implementace SSL/TLS a dalších bezpečnostních opatření.

4.2 Relační databáze

Pro relační databázi bude použita MariaDB[4]. Jedná se o vyspělý open-source relační databázový systém, který nabízí širokou škálu funkcí a vysokou úroveň spolehlivosti.

4.3 Grafová databáze

Pro grafovou databázi bude použit Neo4j[5]. Neo4j je grafová databáze, která umožňuje efektivní ukládání a dotazování nad grafovými daty.

4.3.1 Výhody použití Neo4j:

- **Optimalizace pro grafové dotazy**: Umožňuje rychlé prohledávání grafu, což je ideální pro modelování distribuční sítě.
- **Cypher Query Language**: Pokročilý dotazovací jazyk speciálně navržený pro práci s grafy.
- **Výkon**: Vysoký výkon při zpracování komplexních grafových struktur a vztahů.
- **Vizualizace**: Poskytuje nástroje pro vizualizaci grafových dat, což usnadňuje analýzu a ladění.

4.4 Event Store

Pro `Event Store` bude použita EventStoreDB[6]. EventStoreDB je otevřená databáze navržená specificky pro ukládání událostí v architekturách využívajících Event Sourcing.

4.4.1 Výhody použití EventStoreDB:

- **Škálovatelnost**: Podporuje horizontální škálování pro zvládnutí rostoucího objemu událostí a zátěže systému.
- **Optimalizace pro zápis událostí**: Navržena pro efektivní zápis a ukládání velkého množství sekvenčních událostí, což je ideální pro Event Sourcing.

[3] https://nginx.org/
[4] https://mariadb.org/
[5] https://neo4j.com/
[6] https://www.eventstore.com/

- **Podpora projekcí**: Umožňuje vytvářet projekce dat v reálném čase, což usnadňuje vytváření čtecích modelů a zpracování dotazů.
- **Spolehlivost a konzistence**: Nabízí silnou záruku konzistence dat a odolnost vůči chybám, což je kritické pro systémy vyžadující vysokou integritu dat.
- **Integrace s různými platformami**: Poskytuje API pro více programovacích jazyků a dobře se integruje s existujícími systémy a technologiemi.

4.5 Cache

Pro cache bude použit KeyDB[7], což je open-source in-memory databáze kompatibilní s Redis API[8], ale nabízí některá vylepšení oproti Redis.

4.5.1 Výhody použití KeyDB:

- **Kompatibilita s Redis**: Plná kompatibilita s Redis API usnadňuje integraci a přechod z Redis.
- **Vyšší výkon**: KeyDB podporuje multithreading, což zvyšuje propustnost a snižuje latenci oproti jednovláknovému Redis.
- **Licenční výhody**: KeyDB je vydáván pod Apache 2.0 licencí, což může být výhodné z hlediska právních a licenčních omezení.
- **Pokročilé funkce**: Podpora pro Active-Active replikaci, což zlepšuje dostupnost a konzistenci dat v distribuovaném prostředí.

4.6 Message Queue

Pro message queue bude použit RabbitMQ[9]. RabbitMQ je robustní message broker, který podporuje různé messaging protokoly a vzory.

4.6.1 Výhody použití RabbitMQ:

- **Spolehlivost**: Nabízí funkce jako potvrzení zpráv, trvalé fronty a potvrzení doručení, což zajišťuje spolehlivou komunikaci mezi komponentami.
- **Flexibilita**: Umožňuje složité routingové mechanismy, jako je směrování na základě témat nebo klíčů.
- **Komunita a podpora**: Široká uživatelská základna a množství dostupných knihoven pro různé programovací jazyky usnadňují implementaci a řešení problémů.

[7] https://docs.keydb.dev/
[8] https://redis.io/
[9] https://www.rabbitmq.com/