Masarykova univerzita
Fakulta informatiky

Open source administratívne rozhranie pre Stream Data Platform

Diplomová práca

Bc. Martin Kočiský

Brno, jar 2019

Masarykova univerzita
Fakulta informatiky

Open source administratívne rozhranie pre Stream Data Platform

Diplomová práca

Bc. Martin Kočiský

Brno, jar 2019

Na tomto mieste sa v tlačenej práci nachádza oficiálne podpísané zadanie práce a prehlásenie autora školského diela.

Prehlásenie

Prehlasujem, že táto diplomová práca je mojím pôvodným autorským dielom, ktoré som vypracoval samostatne. Všetky zdroje, pramene a literatúru, ktoré som pri vypracovaní používal alebo z nich čerpal, v práci riadne citujem s uvedením úplného odkazu na príslušný zdroj.

Bc. Martin Kočiský

Vedúci práce: RNDr. Michal Batko, Ph.D.

Poďakovanie

Chcel by som sa poďakovať všetkým, ktorí mi počas písania tejto práce vyjadrili podporu.

Zhrnutie

Cieľom diplomovej práce je analyzovať súčastnú formu administratívneho rozhrania pre Stream Data Platform vyvíjanú firmou Lundegaard, navrhnúť jeho rozdelenie na open source časť pre platformu Apache Kafka a jeho privátne rozšírenie. Následne toto rozdelenie naimplementovať. Ďalej rozšíriť toto administratívne rozhranie o novú funkcionalitu nad Kafka Connect API a výslednú open source aplikáciu publikovať.

Kľúčové slová

open source, administratívne rozhranie, web, analýza, Stream Data Platform, Apache Kafka, JavaScript, React, …

Obsah

1 Úvod

Stream Data Platform Management (SDPM) je administratívne rozhranie navrhnuté firmou Lundegaard určené na správu svojej platformy Stream Data Platform (SDP). Pôvodne bolo naimplementované spôsobom, ktorý neumožňuje jeho publikáciu ako open source projekt.

Pozostáva z rozhrania pre administráciu platformy Apache Kafka a rozhrania pre administráciu privátnej nadstavby nad touto platformou. Apache Kafka je open source platforma určená na spracovanie rýchlych dátových prúdov. Tieto dve časti sú v pôvodnom projekte implementačne previazané. Preto, pokiaľ chceme projekt publikovať ako open source, je potrebné ich od seba oddeliť.

Prvým cieľom tejto diplomovej práce je analyzovať súčasnú štruktúru a implementáciu projektu SDPM, následne navrhnúť architektonické rozdelenie, ktoré oddelí proprietárny kód od kódu zamýšľaného na publikáciu. Open source časť bude pozostávať z administratívneho rozhrania pre platformu Apache Kafka. Privátna časť bude koncipovaná ako rozšírenie nad open source časťou spôsobom, ktorý zároveň umožní správu platformy Stream Data Platform. Dôvodom pre toto rozdelenie je cielená publikácia tohto open source rozhrania.

Druhým cieľom je rozšírenie open source časti o novú funkcionalitu. Kafka Connect je open source framework určený na prepojenie Apache Kafky s externými systémami. Napríklad s databázami, key-value úložiskami, vyhľadávacími indexmi a súborovými systémami.[1] Poskytuje API vo forme REST.

Open source časť tohto administratívneho rozhrania bude rozšírená o možnosť využitia tohto API. Bude pridané webové rozhranie umožňujúce vykonávanie operácií nad týmto API so svojím prepojením na back end.

Ďalsím cieľom je publikácia tohto administratívneho rozhrania. Prvotne bude publikované ako open source projekt na verejne prístupnom repozitári Git. Publikácia na verejnom repozitári umožní prístup k jeho zdrojovým kódom, jeho jednoduchú integráciu a využitie ďalším užívateľom. Týmto sa zvýši prínos už doposiaľ naimplementovaného kódu a novo pridaných funkcionalít. Software publikovaný ako open source, často benefituje z prínosou open source komunity, jej spätnej väzby a z prínosu tretích strán, ktoré sa rozhodli ho integrovať,

rozšíriť alebo využívať vo svojich vlastných systémoch. V súčasnosti je mnoho firiem, ako napríklad Apple[2] alebo Salesforce[3], ktoré využívajú platformu Apache Kafka.

2 Stream Data Platform Management

Stream Data Platform Management je administratívne rozhranie nad platformou Stream Data Platform firmy Lundegaard. Bolo navrhnuté a implementované ako súčasť diplomovej práce Mgr. Jozefa Krcha na Fakulte informatiky Masarykovej univerzity.

Stream Data Platform je platforma určená na spracovanie rýchlych dátových prúdov v reálnom čase, ktoré umožňuje podnikanie založené na udalostiach. Platforma je založená na technológiách Apache Kafka a Apache Kafka Streams. Zdroje dát, produkujúce udalosti, sú sekvenčne logované v žurnáloch pre budúce spracovanie. Toto umožňuje vytvorenie akejsi abstrakcie, ktorá oddeľuje dáta od systémov, ktoré ich budú v budúcnosti spracovávať.[4]

Cieľom tohto administratívneho rozhrania je uľahčiť správu a konfiguráciu systémov tejto platformy. Grafické rozhranie často zjednodušuje administráciu systémov pre užívateľov, ktorí nemusia byť podrobne oboznámení s daným systémom alebo pre užívateľov, ktorí vykonávajú časté operácie nad daným systémom.

2.1 Architektúra

Stream Data Platform Management je navrhnutá ako enterprise aplikácia. Jedným z cieľov pri jej vývoji bolo, aby bola jednoducho nasaditeľná na akomkoľvek systéme. Preto bola integrovaná so službou Docker. Jej dva hlavné moduly boli nakonfigurované ako kontajnery nad ňou. Viac informácií o službe Docker a jej využití v rámci tejto aplikácie nasleduje neskôr.

Aplikácia pozostáva z dvoch hlavných modulov *sdp-admin-be* a *sdp-admin-fe*. Súčasťou je taktiež modul zahŕňajúci niekoľko vedlajších nutných súčastí *commons-pom*. Tento modul zahŕňa napríklad modul *kafka-commons*, ktorý je konfiguračnou nadstavbou nad Apache Kafka.

Moduly pre front end *sdp-admin-fe* a modul pre back end *sdp-admin-be* sú navrhnuté ako samostatné kontajnery pre Docker. V rámci tejto aplikácie sú tieto dva kontajnery spustené cez Docker spolu s kontajnerom pre Apache Kafka a pre Apache Zookeeper.

Celá aplikácia je postavená na službe Maven. Maven umožňuje jej jednoduché zostavenie, konfiguráciu a správu závislostí jej oddelených modulov.

2.1.1 Integrácia s Docker

Docker je program, ktorý umožňuje virtualizáciu na úrovni operačného systému. Každá aplikácia, ktorú chceme spustiť nad službou Docker, musí byť navrhnutá ako samostatný balík alebo kontajner. Každý kontajner obsahuje všetky súčasti danej aplikácie. Je definovaný súborom s názvom *Dockerfile*. Ide o textový súbor obsahujúci konfiguráciu virtuálneho prostredia, inštaláciu všetkých závislostí, príkazy potrebné na zostavenie danej aplikácie a jej následné spustenie.

Docker zoberie kontajner a na základe jeho *Dockerfile* vytvorí spustiteľný obraz. Jedinou nutnosťou pre jeho beh je mať nainštalovaný Docker. Závislosti jednotlivých kontajnerov nie sú závislé na vonkajšom prostredí, a teda nie je nutná žiadna ďalšia konfigurácia. Je to vhodný nástroj na jednoduché spúšťanie aplikácií na akomkoľvek systéme.

V prípade SDPM je aplikácia postavená na niekoľkých kontajneroch. Pôvodne aplikácia pozostáva z piatich kontajnerov. Jeden kontajner obsahuje súčasti nutné pre beh privátnej časti aplikácie. Open source časť bude po rozdelení aplikácie pozostávať zo štyroch Docker kontajnerov.

Prvý kontajner je využitý na beh aplikácie Zookeeper. Je to open source program od Apache Foundation. Slúži na centralizovanú správu konfigurácií, na distribuovanú synchronizáciu a poskytovanie skupinových služieb.[5] Je nutnou súčasťou časti tejto aplikácie.

Druhý kontajner je využitý na beh aplikácie Apache Kafka.

Ďalšie dva kontajnery slúžia na beh jednotlivých častí aplikácie, jeden na beh front endu a druhý na beh back endu. Ich samostatná konfigurácia potenciálne umožňuje ich beh aj oddelene.

Každý z týchto kontajnerov sprostredkuje svoje služby cez REST rozhrania dostupné na rôznych adresách a portoch. Východzie nastavenie pre beh tejto aplikácie je beh na adrese *0.0.0.0*. Tieto parametre je možné nakonfigurovať, a tým lepšie prispôsobiť aplikáciu pre rôzne využitie. Táto konfigurácia je súčasťou súboru *docker-compose.yml*.

2.1.2 Modul *sdp-admin-be*

Modul *sdp-admin-be* je založený na frameworku Spring Boot. Tento framework umožňuje jednoduché vytvorenie samostatne stojacich aplikácií, ktoré nie je nutné komplikovane konfigurovať. Zahŕňa základné súčasti pre svoj beh, ako napríklad webový server.[6]

Táto časť je koncipovaná ako REST rozhranie slúžiace na sprostredkovanie prístupu k službám a dátam z platformy Apache Kafka a Stream Data Platform. Implicitný webový server využitý v Spring Boot je Apache Tomcat. Spring na ňom spúšťa tento modul aplikácie na adrese *http://localhost:8080*. Túto adresu je však možné jednoducho nakonfigurovať a v prípade spustenia ako kontajner na službe Docker je ďalej premapovaná v jeho konfigurácii.

Toto REST rozhranie je navrhnuté a postavené na technológii Spring MVC. Je to originálny webový framework založený na Servlet API[7], ktorý umožňuje využitie anotácií na zjednodušenie implementácie webových služieb. Základnou súčasťou aplikácie sú kontrolery označené pomocou anotácie *@RestController*. Kontrolery využívaju služby *service* tried, ktoré sprostredkujú prístup ku hlavným komponentám.

Detailnejšie informácie o implementácii v tejto časti nie sú nutné. O jednotlivých triedach a konfigurácii tohto modulu uvediem viac v kapitole o rozdelení tohto modulu.

2.1.3 Modul *sdp-admin-fe*

Modul *sdp-admin-fe* je druhým hlavným modulom tejto aplikácie. Slúži ako front end. Je napísaný prevažne v jazyku JavaScript. Pozostáva z veľkého množstva modulov a knižníc. Jeho hlavnými súčasťami, ktoré budú spomenuté, sú napríklad React, Redux a React Router.

React je knižnica určená na implementáciu užívateľských rozhraní. Jej základným cieľom je zostavenie užívateľského rozhrania z komponent, ktoré sú oddelené a zapuzdrujú svoju vlastnú logiku a stav. Vo svojej syntaxi využíva jazyk JavaScript XML (JSX). Na základe tejto štruktúry založenej na komponentách je táto aplikácia implementovaná ako single-page. To znamená, že nie je nutné opakované dotazovanie sa na grafické rozloženie daného webového rozhrania.

Stačí sa dotazovať na zmenu dát, ktoré sa uchovávajú v stave danej aplikácie.

Redux je knižnica, ktorej cieľom je jednoduchá správa stavu aplikácie. Poskytuje stavový kontajner, v ktorom je uložený stav celej aplikácie ako jeden strom. Zmena stavu tohto stromu môže nastať len po vyslaní akcie popisujúcej danú zmenu. Na popis toho, ako tieto akcie menia stav tohto stromu sa využívajú *reducers*.[8]

React Router je knižnica určená na smerovanie na základe webových adries a zobrazovanie správnych komponent. Je postavená na knižnici React. Umožňuje jednoduché pridávanie zobrazení na webe spôsobom, ktorý zaručuje, že URL zodpovedá zobrazeniu užívateľského rozhrania.[9]

V pôvodnom projekte bola využitá verzia React Router 3. Pred verziou 4, React Router bol prevažne statický. Cesty boli deklarované ako súčasť inicializácie aplikácie. Interne premieňal elementy hierarchie <*Route*> na konfiguráciu ciest. [9]

To znamená, že v priebehu zostavenia projektu, React Router vytvorí túto internú konfiguráciu. Na základe dokumentácie a niekoľkých praktických pokusov sme dospeli k záveru, že túto konfiguráciu nie je možné v tejto verzii tejto knižnice dynamicky rozšíriť. Preto sme sa rozhodli nahradiť v tomto projekte túto knižnicu jej novšou verziou. Ďalšie podrobnosti o tejto zmene uvediem neskôr.

React Router verzie 4 prišiel s novým spôsobom pre smerovanie, ktorý viac zodpovedá modelu frameworku React na modelovanie užívateľských rozhraní.

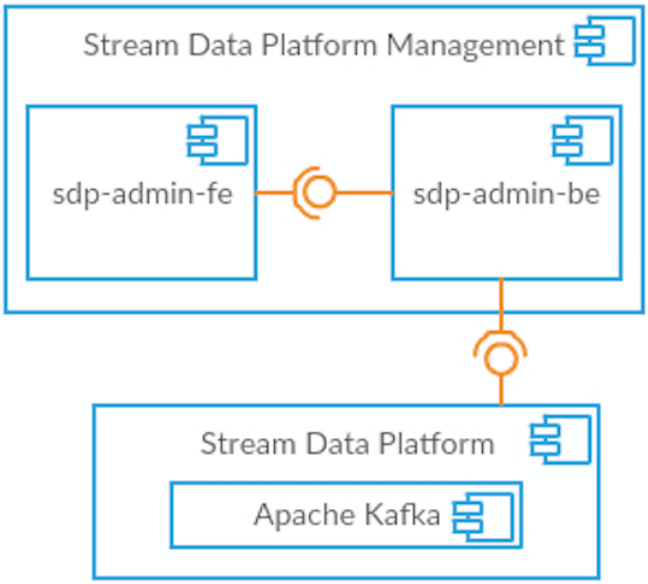
Z dokumentácie vieme nasledovné. Keď hovoríme o dynamickom smerovaní v kontexte React Router 4, myslí sa tým smerovanie, ktoré nastáva v momente vykreslovania aplikácie a nie počas konfigurácie alebo mimo behu aplikácie. To znamená, že všetko v React Router 4 je komponentou frameworku React. Pokiaľ parameter cesty URL nezodpovedá parametru cesty, priradenej danej komponente, React vykreslý *null*.[10]

Vďaka tomuto prístupu je možné dané komponenty neskôr rozšíriť. Nie sú závislé na žiadnej statickej konfigurácií je možné ich meniť do momentu ich vykreslenia alebo ich meniť a znovu vykresliť.

Zostavenie tohto modulu aplikácie je navrhnuté nasledovne. Na správu svojich závislostí využíva tento modul dva frameworky, NPM a Yarn. Na svoje zostavenie využíva Webpack. Pôvodná konfigurácia webpacku pozostáva z troch častí. Kofigurácie pre produkčné zostavenie, konfigurácia pre vývojové zostavenie a ich spoločná časť konfigurácie. Konfigurácia tohto zostavenia sa v rámci tohto projektu čiastočne zmenila a viac o tejto zmene uvediem neskôr.

2.2 Previazanie SDPM A SDP

Stream Data Platform Management je navrhnutá s cieľom umožniť jednoduchú administráciu Stream Data Platform. Pôvodne bola táto platforma spravovaná pomocou skriptov a pomocou netriviálnych príkazov cez terminál. Toto však nebolo pre platformu jej rozsahu vhodné riešenie. Preto bolo navrhnuté webové administratívne rozhranie, ktoré tento proces značne zjednodušilo.



Obr. 2.1: Diagram hlavných komponent aplikácie SDPM a jej pôvodné naviazanie na SDP

Na obrázku 2.1 sa nachádza diagram znázorňujúci hlavné moduly tejto aplikácie. Predstavuje pôvodné architektonické zostavenie tejto

aplikácie. Na diagrame si môžeme všimnúť dve komponenty - SDPM a jeho prepojenie na SDP. Súčasťou SDPM sú dva hlavné moduly, ktoré je nutné rozdeliť a oddeliť tým funkcionalitu určenú pre open source aplikáciu a pre privátne rozšírenie. V neskoršej kapitole uvediem diagram po rozdelení tejto aplikácie a po pridaní rozhrania pre Kafka Connect.

Front endová časť aplikácie prezentuje užívateľovi webové rozhranie umožňujúce jednoduchú vyzualizáciu dát a ich správu. Pri vykonaní bežnej operácie nad dátami platformy je vytvorená REST požiadavka. Základným formátom pre posielanie a prijímanie dát je JSON. Požiadavka je následne poslaná lokálne bežiacej back endovej časti tejto aplikácie.

Back endová časť prijíma dané požiadavky a zaobstaráva ich spracovanie. Na základe zvolenej HTTP metódy a parametrov cesty URL je identifikovaná korektná metóda konektoru určená na jej spracovanie. Daná metóda následne vezme telo požiadavky a cez servisnú vrstvu ju predá vhodnej komponente na ďalšie spracovanie. Následne vráti odpoveď na danú požiadavku vrstve front endu. Aplikácia front endu si dáta uchováva vo svojom stave. Prezentuje ich užívateľovy vo forme niekoľkých webových náhľadov.

3 Apache Kafka

Apache Kafka je open source distribuovaná platforma určená na správu a spracovanie rýchlych prúdov dát. Používa sa na zostavenie tokov v reálnom čase, ktoré dostávajú dáta z aplikácií. Následne na zostavenie aplikácií, ktoré sú schopné spracovať tieto prúdy dát v reálnom čase.[11]

Má päť základných skupín aplikačného programového rozhrania.

- Producer API
- Consumer API
- Streams API
- Connect API
- AdminClient API

V rámci tejto práce sa ďalej budeme venovať Kafka Connect API. Zvyšné skupiny API už boli implementované v tejto aplikácii.

3.1 Existujúce možnosti

Apache Kafka je rozsiahlo využívaná platforma. Preto by sme chceli spomenúť pre ňu niekoľko komerčne prístupných administratívnych rozhraní. Ich častou nevýhodou je ich spoplatnenie v rámci komerčného využitia a iné licenčné obmedzenia.

Jedným takýmto produktom je Kafka Monitoring Suite, ktoré je súčasťou Lenses. Je to softwarové riešenie určené na monitorovanie a správu platformy Apache Kafka. V rámci tohto produktu je integrované s ďalšími službami a je aj spoplatnené.[12]

Ďalším príkladom je Kafka Tool of firmy DB Solo[13] alebo Kafka Manager zverejnený na GitHub firmou Yahoo[14]. Každá ďalšia alternatíva prináša svoje plusy aj mínusy spojené s jej použitím, integráciou a jej ďalším rozšírením v prípade vlastných špecializovaných produktov.

3.2 Kafka Connect

Kafka Connect slúži ako nástroj na škálovateľné a spoľahlivé streamovanie dát medzi Apache Kafka a inými systémami. Umožňuje jednoduché definovanie konektorov, ktoré presúvajú veľké kolekcie dát *z* a *do* Kafky. Môže napríklad presunúť celú databázu dát do Kafky, alebo dáta z Kafky presunúť do nejakého iného uložiska alebo systému, kde dáta môžu byť následne spracované.[1]

REST rozhranie pre Kafka Connect ponúka množinu metód určených na administráciu klastra na ktorom sú vykonávané úlohy daných konektorov. Toto zahŕňa metódy, ktoré sú určené na zobrazovanie konfigurácií konektorov, stav ich úloh (tasks) a ďalšie operácie nad nimi. Tiež zahŕňa metódy na konfiguráciu a validáciu ďalších rozšírení nad Kafka Connect.

3.2.1 Kafka Connect API

Nasleduje zoznam a popis podporovaných REST požiadaviek z dokumentácie rozhrania pre Kafka Connect.[1] Všetky nasledujúce požiadavky sú naimplementované ako súčasť rozšírenia open source časti tejto aplikácie.

`GET /connectors`

Vráti zoznam aktívnych konektorov.

`POST /connectors`

Vytvorí nový konektor. Telo požiadavky musí obsahovať JSON s menom konektoru a parametrami jeho konfigurácie.

`GET /connectors/{name}`

Vráti informácie o konektore s daným menom.

`GET /connectors/{name}/config`

Vráti parametre konfigurácie konektoru s daným menom.

`PUT /connectors/{name}/config`

Vykoná update konfigurácie konektoru s daným menom.

`GET /connectors/{name}/status`

Vráti stav daného konektoru. Obsahuje informácie o jeho behu, ku ktorému workeru je priradený, informáciu o chybe, pokiaľ k nej došlo a stav jeho úloh.

`GET /connectors/{name}/tasks`

Vráti zoznam úloh, ktoré bežia na konektore.

`GET /connectors/{name}/tasks/{taskid}/status`

Vráti stav úlohy s daným ID. Obsahuje informácie o jej behu, ku ktorému workeru bola priradená a v prípade chyby informácie o nej.

`PUT /connectors/{name}/pause`

Pozastaví konektor a jeho úlohy. Zastaví spracovanie správ až do jeho opätovného spustenia.

`PUT /connectors/{name}/resume`

Obnoví beh konektoru v prípade, že je pozastavený.

`POST /connectors/{name}/restart`

Reštartuje konektor.

`POST /connectors/{name}/tasks/{taskId}/restart`

Reštartuje individuálne úlohy na konektore pokiaľ zlyhali.

`DELETE /connectors/{name}`

Vymaže daný konektor.

`GET /connector-plugins`

Vráti zoznam rozšírení nainštalovaných v Kafka Connect klastri.

`PUT /connector-plugins/{co-type}/config/validate`

Validuje poskytnuté hodnoty konfigurácie oproti jej definícii. Vráti doporučené hodnoty alebo chybovú správu.

3.2.2 Popis funkcionality

Každý konektor pozostáva z množiny úloh. Po priradení konektoru do výpočetného klastra, zmene jeho konfigurácie alebo po zmene počtu

jeho úloh, sa vykoná prerozdelenie úloh. Prerozdelenie úloh zaistí, aby mal každý pracovný uzol približne rovnaké množstvo práce.[1]

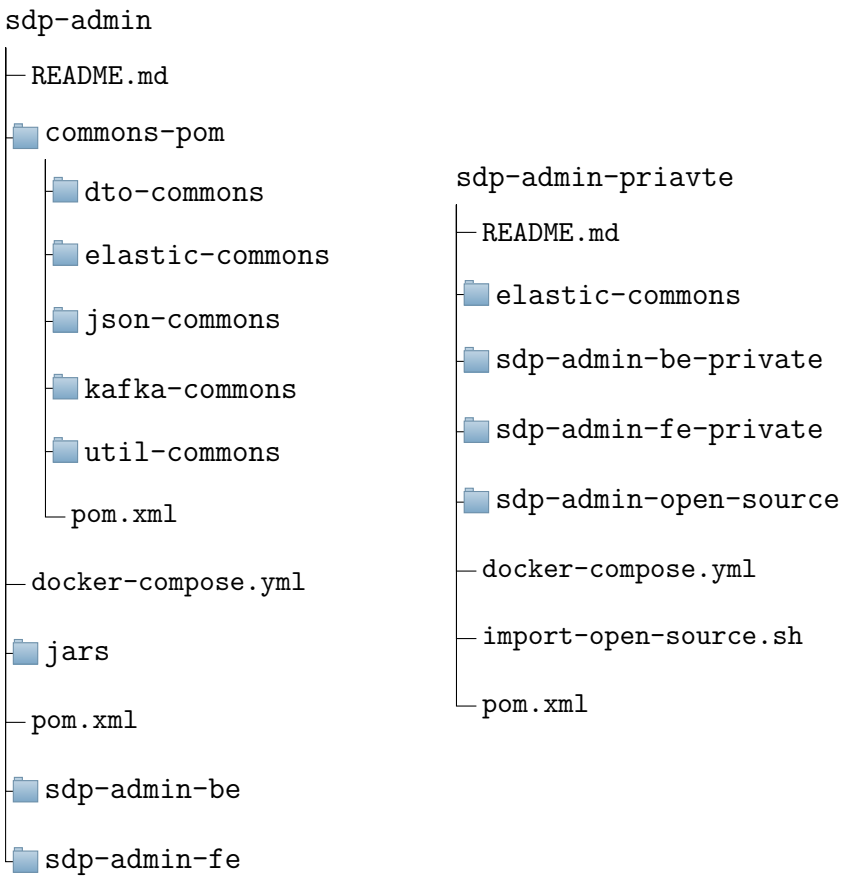
Toto rozhranie je možné využiť na získanie informácií o týchto konektoroch. Jeho stav zahŕňa napríklad informácie o tom, ktoré pracovné uzly spracúvajú ktoré úlohy. O každom konektore si vieme vyžiadať zoznam jeho úloh. Je možné mu tiež priradiť novú konfiguráciu. Každý konektor je možné reštartovať, pozastaviť, obnoviť jeho beh, alebo ho zmazať. O každej úlohe následne vieme zistiť jej konfiguráciu, môžme sa dotazovať na jej stav alebo ju reštartovať.

Pretože konektory a ich úlohy zverejňujú zmenu svojho stavu na zdielaný topic, ku ktorému sa pristupuje asynchrónne, typicky chvíľu trvá, kým je možné pozorovať zmenu daného stavu.[1]

Toto rozhranie predstavuje všetky základné operácie na správu Kafka Connect. Napriek tomu, že bolo navrhnuté ako REST, nebolo k nemu poskytnuté žiadne grafické webové rozhranie. Bez grafického rozhrania sú uživatelia odkázaní na využívanie neprispôsobených aplikácií určených na zostavovanie týchto požiadaviek. Pridaním grafického webového rozhrania, bude užívateľom značne uľahčené využitie tohto rozhrania. Zobrazenie dát bude tiež implementované v jedoduchej čitateľnej podobe.

4 Rozdelenie rozhrania SDPM

Rozdelenie SDPM rozhrania na open source aplikáciu a jej privátne rozšírenie si vyžadovalo množstvo zmien v oboch hlavných moduloch tejto aplikácie a zároveň aj v konfigurácii pre Maven a pre Docker. Na nasledujúcom diagrame znázorňujem celkovú zmenu adresárovej štruktúry danej aplikácie.



Obr. 4.1: Zmena adresárovej štruktúry. Naľavo je pôvodná a napravo je nová štruktúra celej aplikácie.

Open source aplikácia, ktorá je určená na publikáciu a je súčasťou prílohy tejto práce zodpovedá *sdp-admin-open-source*.

V privátnej časti aplikácie nastali značné zmeny v konfigurácii pre Maven. Bola vytvorená nová konfigurácia na základe pôvodnej. Bolo potrebné prejsť všetky závislosti pôvodnej aplikácie a oddeliť závislosti, ktoré sú určené len pre jej privátnu časť. Následne bolo potrebné spraviť zmeny v konfiguračných súboroch Maven*pom.xml* open source aj privátnej aplikácie tak, aby odzrkadlili zmeny v názvoch jednotlivých modulov a zmeny ich polohy v rámci štruktúry celej aplikácie.

Zmeny v konfigurácii pre open source aplikáciu boli menšie. Nastali zmeny v konfigurácií nutnej pre zostavenie svojich dvoch hlavných modulov *sdp-admin-be* a *sdp-admin-fe*.

V konfigurácii pre Docker bol oddelený jeden kontajner využitý len v privátnej časti. Ďalej, konfigurácia kontajnerov bola upravená tak, aby boli spúšťané ich lokálne obrazy.

Privátna časť aplikácie bola ďalej rozšírená o skript *import-open-source.sh*. Tento súbor obsahuje postupnosť príkazov na zostavenie open source aplikácie spôsobom, ktorý umožňuje import jej častí privátnou aplikáciou. Následne importuje tieto zostavené časti pre Maven a umiestňuje novo vytvorenú knižnicu pre front end do privátneho modulu. Ukážku časti tohto skriptu uvediem v nasledujúcej sekcii.

4.1 Rozdelenie modulu sdp-admin-be

Back end SDPM aplikácie je napísaný prevažne v jazyku Java. Pozostáva z viac ako štyridsiatich tried. Aplikácia ako celok neobsahuje žiadnu dokumentáciu k svojim triedam. Ako prvé bolo nutné prejsť celú aplikáciu po jednotlivých triedach od svojho vstupného bodu a identifikovať, ktoré komponenty sú súčasťou rozhrania pre Apache Kafka, a teda vhodné na zverejnenie.

Vstupný bod aplikácie je hlavná trieda *Application*, ktorá je označená anotáciou *@SpringBootApplication*. Táto trieda vytvára kontext Spring aplikácie a následne ju spúšťa. Privátna aplikácia obsahuje podobnú triedu rozšírenú o ďalšiu Spring anotáciu *@ComponentScan*. Táto anotácia špecifikuje balík (basePackage) open source aplikácie. V tomto balíku Spring automaticky vyhľadá komponenty označené anotáciami a automaticky ich pridá do privátnej aplikácie.

Pôvodná aplikácia obsahuje štyri triedy obsahujúce konfigurácie pre jej služby a päť tried s ich vlastnosťami (properties). Trieda spravujúca konfiguráciu celej aplikácie pre open source bola upravená odobratím privátnych častí. Konfigurácia pre komponenty určené na publikáciu je ponechaná v nezmenenom stave. Trieda spravujúca konfiguráciu v privátnej aplikácií, bola rozšírená o vhodný import open source komponent a svojou formou sa podobá originálu.

Aplikácia ďalej obsahuje päť kontrolerov, ktoré cez svoje REST rozhranie sprostredkujú služby front endu. Do open source aplikácie boli zahrnuté tri z nich, bez nutnosti zmien. Tieto boli *GroupController*, *TopicController* a *DefaultExceptionHandler*. Jeden kontroler bol zahrnutý len do privátnej aplikácie. Posledný kontroler *AdminController* obsahoval metódy, ktoré sú súčasťou rozhrania v open source časti a zároveň metódy, ktoré sú súčasťou privátnej aplikácie a publikované nebudú.

Privátne metódy tohto kontroleru boli oddelené a umiestnené do novej triedy privátnej aplikácie. Nový kontroler v privátnej aplikácii poskytuje rovnaké REST API ako pôvodná aplikácia. Napriek rovnakému mapovaniu prvej časti cesty URL na danú triedu, všetky požiadavky sú svojimi ďalšími časťami cesty unikátne identifikovateľné. Preto je ich Spring schopný správne namapovať aj pri použití oboch týchto tried.

Kontroleri využívajú služby piatich *Service* tried aj s ich rozhraniami. Tri triedy ako v predchádzajúcom prípade, boli zahrnuté do open source aplikácie bez zmeny. Jedna bola zahrnutá len do privátnej aplikácie. Trieda *AdminServiceImpl* si vyžadovala oddeliť svoje metódy. Publikovaná časť obsahuje len dve jednoduché metódy. V privátnej časti bolo nutné upraviť konštruktor a vhodne upraviť metódy využívajúce funkcionalitu z open source časti.

Service využívajú API zo štyroch ďalších komponent, ktoré boli celé zahrnuté do open source. Oddelená časť z *AdminServiceImpl* využíva metódy viacerých komponent, ktoré boli zahrnuté do privátnej časti aplikácie a nebolo potrebné vykonať žiadnu ďalšiu úpravu.

Aplikácia využíva tiež množstvo *Data Transfer Object*, skrátene DTO, ktoré bolo nutné taktiež oddeliť.

4.2 Rozšírenie open source modulu sdp-admin-be

Open source časť, zodpovedajúcu tomuto modulu aplikácie, bolo nutné vhodne oddeliť a odobrať všetky privátne časti aplikácie. Privátna časť, ktorá obsahuje implementovanú logiku na správu platformy SDP, vyžadovala najviac zmien.

Konfiguračný súbor *pom.xml* pre Maven bol upravený a všetky závislosti využívané v privátnej časti aplikácie boli odstránené. Bola pridaná špecifikácia, akým spôsobom má byť aplikácia zabalená po zostavení - <*packaging*>*jar*</*packaging*>.

Toto bolo potrebné, pretože sa aplikácia zostavuje spolu s konfiguráciou a obraz pre Docker. Zo zostaveného *jar* pôvodnej aplikácie nebolo možné importovať žiadne triedy. Aby bolo možné importovať súčasti tejto open source aplikácie, z privátnej aplikácie, bolo nutné zostaviť aplikáciu do *jar* formátu predtým, ako sa zostaví pomocou rozšírenia pre Docker. Výsledkom sú dva *jar* balíky, *sdp-admin-be.jar.original* a *sdp-admin-be.jar*. Pôvodný balík je ten, z ktorého je možné robiť import tried.

Nasleduje ukážka skriptu *import-open-source.sh*, ktorý slúži na import z open source aplikácie do privátnej aplikácie.

```
cd ./sdp-admin-open-source/sdp-admin-be
mvn install
...
mv ./sdp-admin-open-source/sdp-admin-be/target/\
    sdp-admin-be.jar.original .
mv sdp-admin-be.jar.original sdp-admin-be.jar
...
mvn org.apache.maven.plugins:\
maven-install-plugin:3.0.0-M1:install-file \
-Dfile=sdp-admin-be.jar \
-DgroupId=eu.lundegaard.sdp \
-DartifactId=sdp-admin-be \
-Dversion=0.50.3-SNAPSHOT \
-Dpackaging=jar
```

Listing 4.1: Import open source časti aplikácie

Plugin v príkaze *mvn* vo vyššie uvedenom skripte je určený na lokálny import závislosti cez Maven. Po konzultácii s firmou Lundegaard bude aplikácia iniciálne publikovaná ako celý projekt na Git. V prípade, že by sa časti tejto aplikácie publikovali oddelene, napríklad tento modul pre back end na repozitári Maven, bolo by možné tento lokálny import vynechať a importovať daný balík typickým spôsobom.

4.3 Rozdelenie modulu sdp-admin-fe

4.3.1 Cieľ výsledného rozdelenia

Front end SDPM pozostáva z množstva komponent, ich kontajnerov a funkcií, predstavujúcich rôzne časti webového rozhrania tejto aplikácie. Aplikácia bola postavená na viacerých knižniciach, ako napríklad React, Redux a React Router. Na správu závislostí boli použité NPM a Yarn. Na zostavenie aplikácie Webpack. Pretože tento modul bol značne previazaný s radom ďalších knižníc, bolo nutné zvážiť možnosti rozšírenia tejto aplikácie. Jednou z hlavných požiadaviek na privátne rozšírenie tejto aplikácie bolo zachovanie pôvodnej funkcionality.

Dôležitým cieľom preto bolo minimalizovať potrebný zásah do už existujúceho kódu aplikácie. Snažili sme sa docieliť, aby privátna aplikácia bola maximálne nezávislá na zmenách v open source aplikácii. Ďalším cieľom bolo rozdeliť túto aplikáciu spôsobom, ktorý by umožnil čo najjednoduchšiu integráciu open source a privátnej aplikácie do jedného celku.

Pôvodná aplikácia nebola dizajnovaná na to, aby ju bolo možné zahrnúť ako celok do inej aplikácie, alebo aby ju bolo možné jednoducho rozšíriť.

Zvážme jeden možný prístup. Zahrnutie celej aplikácie do privátnej. To by vyžadovalo vytvoriť aplikáciu, ktorá obsahuje množstvo rovnakej logiky. Ďalej by privátna aplikácia musela ako jedno zo svojich webových zobrazení sprístupniť celú open source aplikáciu bez zmeny. Jedno zo zobrazení privátnej aplikácie by vykreslovalo celú *<Root>* komponentu open source aplikácie. To by znamenalo, že by bolo duplikované menu a nebolo by možné znovu použiť žiadnu časť open source implementácie bez jej duplikácie.

Aplikácie by v podstate boli dve. Open source aplikácia by len bola nie veľmi esteticky zahrnutá do tej privátnej. Preto si myslíme a naším návrhom je, že lepšou možnosťou bude sprístupniť časti open source aplikácie a tým umožniť ich modifikáciu a rozšírenie.

4.3.2 Problémy pred rozdelením

Prvým z problémov, ktorý sa v priebehu tohto projektu vyskytol, bolo nájdenie spôsobu, ktorým by bolo možné zahrnúť časti open source aplikácie do privátnej aplikácie. Pôvodná aplikácia využívala na svoje zostavenie Webpack. Ako súčasť svojej konfigurácie, Webpack umožňoval zostavenie v dvoch módoch. Prvým bol produkčný mód a druhým bol developerský mód. Oba tieto módy boli čiastočne závislé na množine spoločných konfiguračných parametrov. Ich súšasťou boli aj parametre popisujúce spôsob optimalizácie aplikácie. Túto konfiguráciu si môžeme všimnúť v súboroch *webpack.prod.js*, *webpack.dev.js* a *webpack.common.js*.

Aplikácia ako celok, bola optimalizovaná tak, že knižničné node moduly a všetky ďalšie závislosti boli zostavené do samostatného súboru. Triedy tejto aplikácie boli zostavené do ďalšieho samostatného súboru, ktorý bol závislí na tom predchádzajúcom. Súbor zostavený z knižníc bol určený na dlhodobé uchovanie ako cache u užívateľa. Pri každom ďalšom zobrazení tejto webovej aplikácie, sa zo serveru obnoví len zostavený súbor, ktorý obsahuje časti našej aplikácie. Tento spôsob optimalizácie je vhodný, pretože pri vývoji tejto aplikácie sa najčastejšie predpokladajú zmeny len v triedach tejto aplikácie a nie v jej závislostiach.

Táto časť konfigurácie popisujúca optimalizáciu, bola presunutá do konfigurácie pre produkčný mód. V prípade vývojového módu, po novom aplikácia nie je týmto spôsobom optimalizovaná. Do konfigurácie vývojového módu bola pridaná nasledujúca konfigurácia pre zostavenie aplikácie.

Táto časť konfigurácie zabezpečuje vytvorenie JavaScript knižnice, z ktorej je ďalej možné importovať komponenty danej aplikácie bežným spôsobom. V prípade privátnej aplikácie je teda open source aplikácia zostavená vo vývojovom móde. Následne je poskytnutá táto knižnica v konfigurácií privátnej aplikácie ako typická *NPM* lokálna

```
output: {
    path: commonPaths.outputPath,
    publicPath: '',
    filename: 'sdp-admin-fe.js',
    library: 'sdp-admin-fe',
    libraryTarget: 'umd',
}
```

Listing 4.2: Rozšírenie konfigurácie určené pre vytvorenie importovateľnej knižnice

závislosť. Privátna aplikácia následne môže po importe všetkých závislostí, importovať všetky korektne exportované objekty z tejto knižnice.

Ďalší z problémov, na ktoré sme narazili, bola nutnosť výmeny knižnice React Router verzie 3, za jej novšiu a funkčne odlišnú verziu 4. Táto zmena si vyžadovala značný zásah do už implementovaného kódu aplikácie. Pôvodná knižnica *react-router* bola tiež naviazaná na knižnicu *Redux*. To si vyžadovalo ďalšie zmeny aj v použití tohto systému.

React Router verzie 3 ktorý bol použitý v pôvodnom projekte obsahoval hlavnú komponentu <*Router*>. Tá obsahovala komponenty <*Route*>, ktoré popisovali parametre cesty a komponenty k nim priradené. Ako som už v predchádzajúcej kapitole uviedol, tieto komponenty sa však nesprávali ako vykreslované komponenty z React. Po zostavení aplikácie sa správali ako statická konfigurácia tohto systému.

V prípade React Router verzie 4, sa fungovanie tejto knižnice značne zmenilo a jej komponenty boli naimplementované ako komponenty z React, teda sú dynamicky vykreslované v rámci aplikácie.

4.3.3 Implementačné zmeny

Všetky komponenty, ich kontajnery, akcie, súbory popisujúce ich konštanty a vlastnosti bolo potrebné postupne prejsť a identifikovať ich náležitosť buď pre open source aplikáciu, alebo jej rozšírenie. Neprítomnosť ich dokumentácie si vyžadovala pre každú komponentu prejsť a identifikovať spätne jej závislosti.

V pôvodnej aplikácií sa nachádzalo okolo tridsiatich komponent. Veľa z nich bolo zachovaných v open source aplikácii. Všetky privátne komponenty a ich súčasti z vyššie menovaných častí tejto aplikácie boli oddelené do nového privátneho projektu. V rozsahu viditeľného rozhrania aplikácie to boli dve zo šiestich položiek v navigačnom menu. Tieto časti boli určené na správu platformy firmy Lundegaard, zobrazovanie dát, informácií a operácií nad nimi.

Komponenta *NavigationMenu* predstavuje ľavé hlavné menu tejto aplikácie. Konfigurácia jeho položiek je umiestnená v rámci tohto súboru ako statická konfigurácia. Do tejto konfigurácie bola pridaná položka spolu s jej ďalšou špecifikáciou pre Kafka Connect. O implementácii tejto komponenty uvediem viac informácií v samostatnej kapitole. Konfigurácia bola upravená spôsobom, ktorý umožnil jej export z tejto aplikácie.

Pôvodná aplikácia si uchovávala svoju históriu vo svojom stave. Knižnica *react-router-redux*, ktorá na to bola použitá, bola z projektu odstránená. Táto knižnica už nie je udržovaná. Pôvodne bola navrhnutá ako doplnok k React Router pred verziou 4. Každý výskyt jej použitia v projekte bol upravený, tak aby odzrkadlil povýšenie verzie React Router a jeho novú funkcionalitu.

Po novom si aplikácia uchováva históriu v internom objekte na to určenom. Táto zmena si vyžiadala zmenu kódu, ktorý spravoval jej zapisovanie a jej čítanie z daného objektu. Ďalšie úpravy v implementácii týkajúce sa tejto zmeny nastali v *reducers* pre Redux.

Nasledujú zmeny, ktoré vyplývajú z použitia knižnice Redux a implementácie jej použitia. Rozsiahle zmeny nastali v súbore akcií, ktorý obsahuje metódy určené na popis zmeny stavu aplikácie. Tieto metódy menia stav pri priechode aplikáciou, ktorý je uložený v *store*. V prípade zmeny zobrazenia komponent spravujú tiež REST požiadavky na back end a zapisujú zmeny stavu zobrazených dát. Všetky akcie bolo potrebné postupne prejsť a oddeliť tie, určené pre open source aplikáciu a tie, pre privátne rozšírenie.

Ďalšie väčšie zmeny nastali v súboroch popisujúcich *reducers*. To sú zoskupenia funkcií vykonávajúce vyššie spomenuté akcie na zmenu stavu tejto aplikácie. Tak ako predtým ich bolo potrebné oddeliť a implementačne upraviť, tak aby ich bolo možné ďalej rozšíriť. V rámci týchto zmien boli vytvorené objekty, ktoré boli vyexportované tak, ako v prípade navigačného menu ako *"named export"*. Exporty s me-

nom identifikujú konkrétne objekty, a narozdiel od *"default export"*, je možné ich mať viac v rámci súboru alebo aplikácie.

Do open source aplikácie bola pridaná funkcia *openSourceInit*. Táto funkcia po novom zahŕňa inicializáciu aplikácie. Jej parametrami sú *rootReducer* a *history*. Vracia pole konfiguračných parametrov určených pre komponentu Root. Prvým parametrom je objekt *store* určený pre *Redux* a druhým objekt *history* určený pre *React Router*.

Po výmene potrebných knižníc spomenutých vyššie a refaktorizácii kódu spojenou s ich výmenou, je ďalším krokom návrh spôsobu, ktorým bude možné rozšíriť komponenty open source aplikácie a využiť ich v privátnej aplikácii.

4.4 Rozšírenie open source modulu sdp-admin-fe

Vstupným bodom open source aplikácie je súbor *index.js*. Obsahuje inicializáciu a kód, ktorý vykresluje *Root* komponentu. Inicializácia zahŕňa vytvorenie *store* určeného na uchovávanie stavu aplikácie a vytvorenie objektu histórie. Táto inicializácia bola umiestnená do jednej novej metódy, ktorá vracia tieto parametre, ktoré sú následne predané *Root* komponente.

Ďalšou novou súčasťou, ktorá je určená na rozšírenie tejto aplikácie je nasledujúci export.

```
export {
  Root, openSourceInit, menuItems,
  entitiesReducers, uiReducers, history,
};
```

Listing 4.3: Export hlavných častí aplikácie

Toto je *"named export"* hlavných častí aplikácie, ktoré sú základom pre jej externé rozšírenie.

4.4.1 Architektúra rozšírenia

V privátnej aplikácií slúžiacej ako rozšírenie nad open source aplikáciou je možné postupovať nasledovane. Je zrejmé, že ku všetkým

týmto častiam je možné po korektnom zostavení a naimportovaní tohoto projektu pristupovať.

Objekt *menuItems* je množina položiek hlavného menu. Cez tento objekt je možné pridať jeho ďalšie položky. Podobne pre objekty *entitiesReducers* a *uiReducers* ich je možné rozšíriť o akcie ktoré sú využité v privátnych komponentách. Tieto je následne potrebné spojiť rovnakým spôsobom ako v pôvodnej aplikácií. Po ich predaní inicializačnej metóde *openSourceInit* spolu s objektom histórie, ich táto metóda vráti inicializované na predanie komponente *Root*.

4.4.2 Komponenta <Root>

Komponenta *Root* je hlavnou vstupnou komponentou, ktorú React vykresluje. Predstavuje open source aplikáciu ako celok z hľadiska užívateľského rozhrania a internej funkcionality. Boli do nej pridané nasledovné nové parametre.

Prvým novým parametrom je objekt *history*. Jeho funkcionalita sa po výmene React Router čiastočne zmenila. Tak, ako aj v pôvodnej aplikácií, tento objekt obsahuje históriu tejto webovej aplikácie. Nie je však objektom danej knižnice, ale je objektom aplikácie, ktorý sprostredkuje objekt z danej knižnice. Tento parameter bolo potrebné pridať z dôvodu zachovania synchronizácie histórie pôvodnej a rozširujúcej <*Router*> komponenty.

Druhým pridaným parametrom je *extension*. Tento parameter je určený na rozšírenie komponenty <*Router*>, tak ako je znázorňené v nasledujúcej ukážke.

Očakávaným objektom rozšírenia je nová <*Router*> komponenta, ktorá ako svoj parameter *history* dostala rovnaký objekt ako komponenta <*Root*>.

Výsledkom sú dve vnorené <*Router*> komponenty s rovnakou históriou. Pri testovanej implementácii nebolo možné pridať ako rozšírenie len blok ciest <*Route*>, pretože kontrola pri kompilácii si vynucovala, aby blok týchto ciest v privátnej aplikácií bol tiež obalený v komponente <*Router*>.

Korektným rozšírením všetkých týchto komponent, je dosiahnutý nasledujúci stav. Výsledné webové rozhranie je pre užívateľa vizuálnou a funkčnou zhodou s originálnou aplikáciou. Položky hlavného

```
const Root
= ({ store, locale, history, extension }) => (
  <Provider store={store}>
    <IntlProvider locale={locale}
                  messages={translation.en}>
      <Router history={history}>
        <App>
          <Switch>
            <Route exact path={url.KAFKACONNECT}
            component={PageKafkaConnect} />
            { extension }
            <Route exact path={url.ANY}
            component={PageNotFound} />
          </Switch>
        </App>
      </Router>
    </IntlProvider>
  </Provider>
);
```

Listing 4.4: Skrátený vzor Root komponenty z open source aplikácie

menu sú v rozdielnom poradí, ale v prípade záujmu je možné ich zoradiť iným spôsobom.

4.5 Nová architektúra aplikácie

Aplikácia bola rozdelená na open source aplikáciu, ktorá bola publikovaná, a na privátnu nadstavbu nad ňou. Open source aplikácia je úplne nezávislá na akýchkoľvek knižniciach a privátnom kóde firmy Lundegaard, ktorý bol súčasťou pôvodnej aplikácie. Celkovú architektonickú štruktúru tejto aplikácie si môžeme pozreť na nasledovnom diagrame.

Open source aplikácia je v rámci tohto diagramu znázornená komponentami označenými ako open source a komponentami Kafka Connect a Apache Kafka.

```
const Extension = ({ history }) => (
    <Router history={history}>
        <Switch>
            <Route exact path={url.UrlOne}
            component={ComponentOne} />
            <Route exact path={url.UrlTwo}
            component={ComponentTwo} />
            <Route exact path={url.UrlThree}
            component={ComponentThree} />
        </Switch>
    </Router>
);
```

Listing 4.5: Skrátený vzor Extension komponenty vhodnej ako rozšírenie pre Root

5 Nové rozhranie pre Kafka Connect

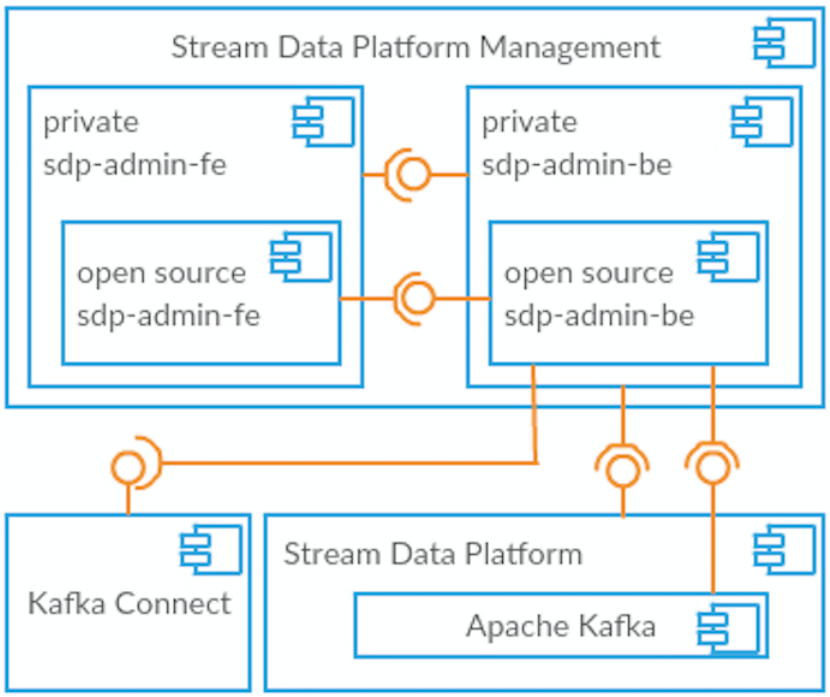
5.1 Back end

Kafka Connect API je navrhnuté ako množina REST dotazov na danú službu. Táto služba a jej rozhranie ako také nemusí byť nutne prístupné na rovnakej adrese ako Apache Kafka. Preto je možné jednoducho nakonfigurovať adresu, kde bude k nemu táto aplikácia pristupovať.

Back end tohto administratívneho rozhrania bol rozšírený o triedu *KafkaConnectController*. Táto trieda je označená pomocou Spring anotácie *@RestController*. Jeho metódy sú naimplementované typickým spôsobom, ktorý zahrňuje popis požiadaviek anotáciami a obsahuje logovanie a dokumentáciu.

Metódy preberajú REST požiadavky z front endu tejto aplikácie. Následne sprostredkujú ich doručenie na adresu, kde je k dispozícií Kafka Connect API. Toto API rozhranie môže byť umiestnené na ľubovoľnom inom serveri.

Hlavnou metódou na to určenou je *restProxyForward*. Táto metóda je základom tohto rozhrania a prijíma nasledujúce parametre.



Obr. 4.2: Diagram komponent aplikácie SDPM, jej naviazanie na SDP a Kafka Connect po rozdelení aplikácie

Parameter, ktorým je predaná pôvodná HTML požiadavka, obsahuje pôvodnú adresu, na ktorú bola požiadavka poslaná, parametre cesty a príslušnú HTML metódu (GET, POST, ...). Ďalším parametrom je jej pôvodný obsah tela.

Metóda zostaví novú URL adresu s pôvodnou cestou. Na novú adresu prepošle telo pôvodnej požiadavky a obsah odpovede predá ako odpoveď na pôvodnú požiadavku pre tento back end.

Celý kód spomínanej triedy je možné nájsť v prílohe tejto diplomovej práce.

5.2 Front end a jeho komponenty

Front end pre Kafka Connect sme napísali nad knižnicou React. Následne sme ho zapracovali do open source časti aplikácie a prispôsobili

použitej grafickej knižnici určenej pre stavbu užívateľských rozhraní *Ant Design - antd*.

Webové rozhranie na administráciu Kafka Connect pozostáva z približne desiatich komponent.

5.2.1 KafkaConnect

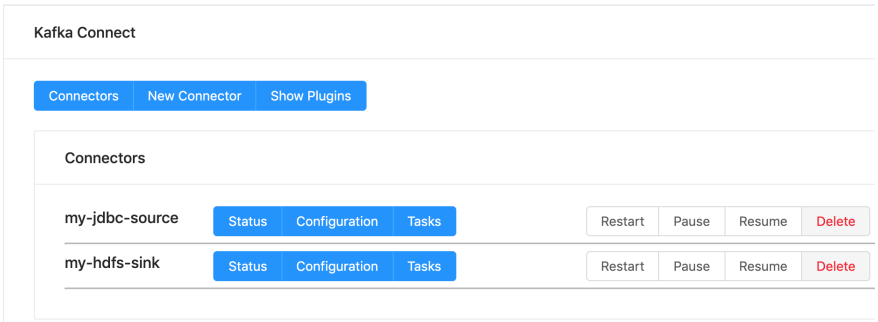
KafkaConnect je hlavnou komponentou tohto rozhrania. Jej hlavná funkcionalita spočíva v zobrazení a navigácií medzi tromi základnými súčasťami tohto rozhrania.

Obsahuje navigačné menu s tromi tlačidlami, ktoré prepínajú medzi zobrazením troch hlavných komponent *KafkaConnectorList*, *KafkaNewConnector* a *KafkaConnectorPlugins*.

5.2.2 KafkaConnectorList

KafkaConnectorList je komponenta, ktorá je jedným z troch základných zobrazení v *KafkaConnect*. Jej hlavnou fukncIou je zobrazenie zoznamu konektorov a ich správa.

Pri zobrazení komponenty sa vyšle žiadosť o dáta ktoré popisujú zoznam konektorov. Následne sú tieto dáta namapované na komponenty *KafkaConnectorListEntry* pre ďalšie zobrazenie.



Obr. 5.1: Ukážka z užívateľského rozhrania KafkaConnect a KafkaConnectorList

5.2.3 KafkaConnectorListEntry

KafkaConnectorListEntry je komponenta, ktorá predstavuje jednu položku zoznamu konektorov. Spravuje jeho dáta a zastrešuje fukncie a zobrazenie komponent nad konkrétnym konektorom.

Zobrazuje meno konektoru a funkcie v podobe tlačidiel, ktoré je možné nad ním zavolať. Základné funkcie, ktoré menia stav konektoru, ale nie zobrazenie webového rozhrania sú *Restart*, *Pause*, *Resume* a *Delete*. Každá z týchto funkcií vráti svoj HTTP Response Code a správu pre uživateľa.

Každý konektor taktiež podporuje zobrazenie svojho, stavu, konfigurácie a zoznamu úloh nad ním bežiacich. Tieto komponenty sú zobrazené ako súčasť komponenty *KafkaConnectorListEntry*, po stlačení príslušného tlačidla. Každé stlačenie niektorého z týchto tlačidel vytvorí nový dotaz na obnovenie dát.

5.2.4 KafkaConnectorConfig

KafkaConnectorConfig je komponenta určená na modifikáciu konfigurácie konektoru. Obsahuje textové pole do ktorého je možné zadať novú konfiguráciu konektoru a tlačidlo na jej potvrdenie. Odpoveď po vykonaní tohto dotazu je zobrazená opätovne v textovom poli.

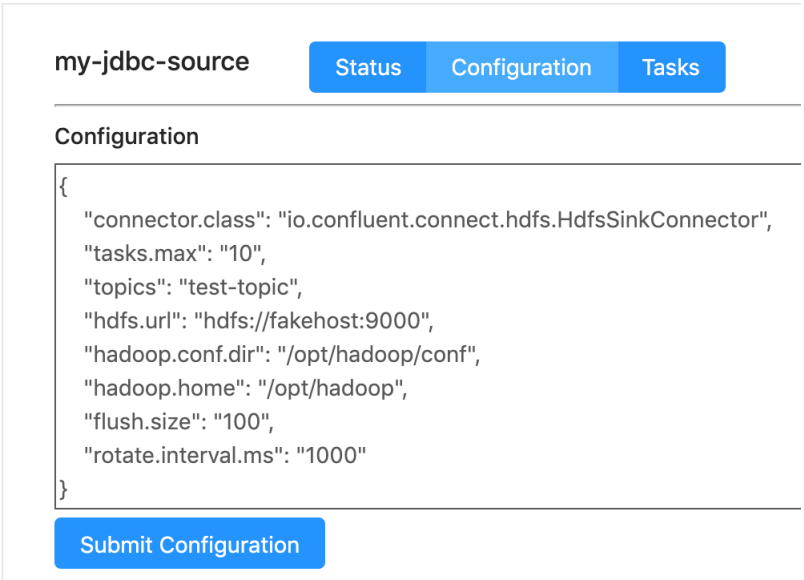
5.2.5 KafkaConnectorStatus

KafkaConnectorStatus je komponenta určená na zobrazenie stavu daného konektoru. Stav je zobrazený ako JSON v needitovateľnom textovom poli.

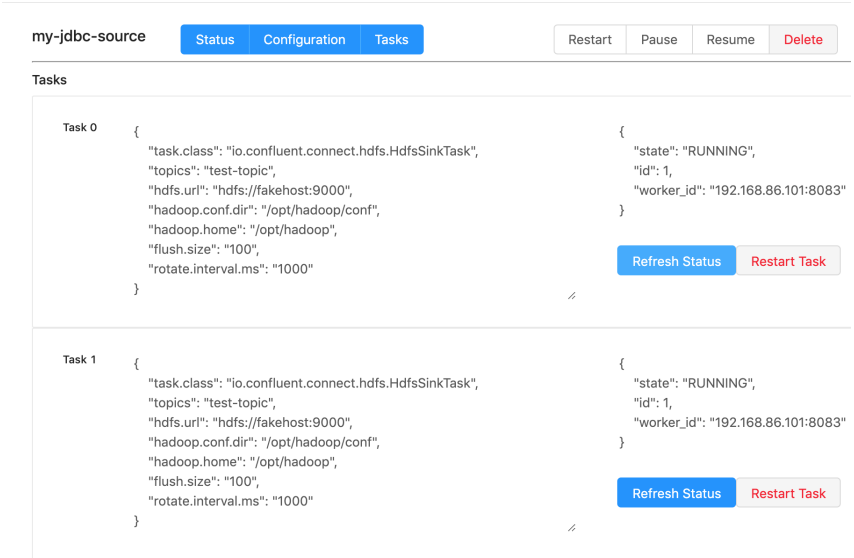
5.2.6 KafkaConnectorTaskList

KafkaConnectorTaskList je komponenta určená na zobrazenie zoznamu úloh nad daným konektorom a na ich správu.

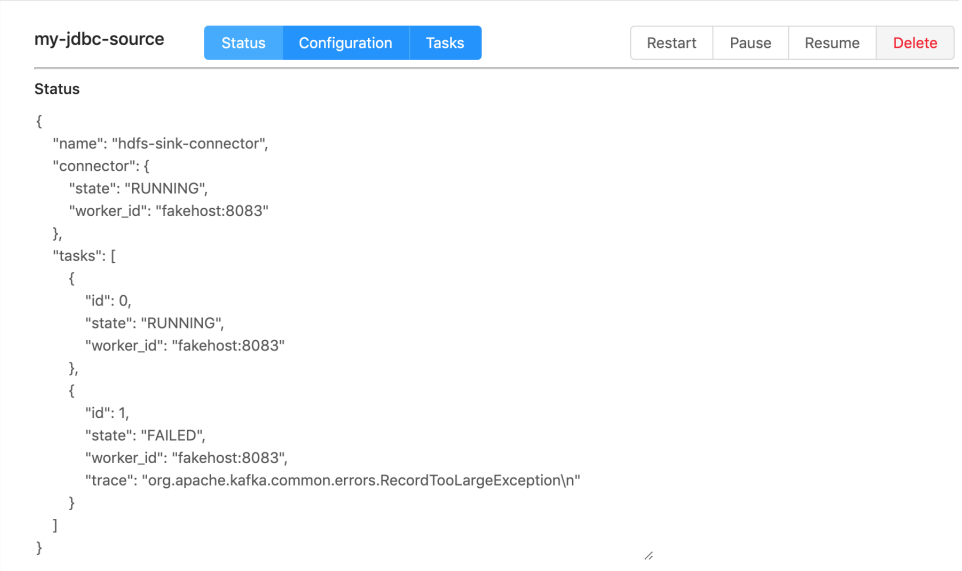
Pri vytvorení komponenta vyšle požiadavku nad daným konektorom o zoznam úloh nad ním. Následne dáta spracuje a namapuje na komponenty KafkaConnectorTaskListEntry.



Obr. 5.2: Ukážka z užívateľského rozhrania KafkaConnectorConfig



Obr. 5.4: Ukážka z užívateľského rozhrania KafkaConnectorTaskList a KafkaConnectorTaskListEntry



Obr. 5.3: Ukážka z uživateľského rozhrania KafkaConnectorStatus

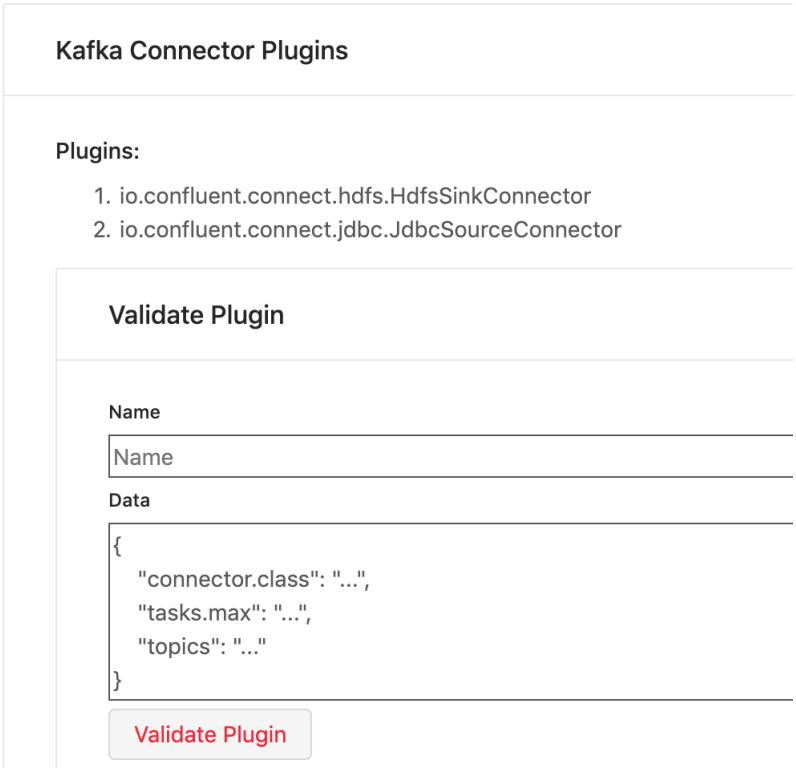
5.2.7 KafkaConnectorTaskListEntry

KafkaConnectorTaskListEntry je komponenta, ktorá predstavuje jednu úlohu na konektore a zobrazuje informácie o nej. Súčaťou informácií o úlohe je index úlohy a jej popis vo formáte JSON. Taktiež komponenta sprostredkuje vyžiadanie stavu úlohy a možnosť jej reštartovania.

Možnosť vyžiadania stavu úlohy a jej reštartovania je užívateľovi poskytnutá vo forme dvoch tlačidiel a textového poľa, ktoré zobrazí informácie o jej stave.

5.2.8 KafkaConnectorPlugins

KafkaConnectorPlugins je komponenta zobrazujúca informácie o rozšireniach na Kafka Connect klastri. Umožňuje taktiež validáciu nových konfigurácií pre konektory. Ako výsledok tejto operácie vráti doporučené hodnoty a ďalšie chybové hlásenia o validovanej konfigurácií.



Obr. 5.5: Ukážka z užívateľského rozhrania KafkaConnectorPlugins

5.2.9 KafkaNewConnector

KafkaNewConnector je komponenta umožňujúca vytvorenie nového konektoru v Kafka Connect klastri. Ako vstup vyžaduje konfiguráciu daného konektoru.

5.2.10 KafkaConnectREST

KafkaConnectREST je množina funkcií, ktoré spravujú REST dotazy a ich odpovede pre všetky vyššie uvedené komponenty a ich súčasti.

6 Publikácia

Stream Data Platform Management je rozsiahla aplikácia. Je postavená z viacerých modulov, ktoré sú napísané prevažne v Jave, JaveScripte a HTML. Je postavená na množine rôznych frameworkov, knižníc a závislostí tak, ako na strane back endu, tak aj na strane front endu. Je komplexne previazaná pomocou REST rozhraní a je naviazaná na služby Apache Kafka, Apache Connect a Zookeeper. Preto bolo pri publikácií, ako open source, dôležité pamätať na niekoľko faktorov ovplyvňujúcich jej budúce použitie.

Prvým faktorom je cieľová skupina užívateľov. V prípade tejto aplikácie sú cieľovou skupinou prevažne užívatelia a firmy, ktoré využívajú software Apache Kafka. Preto môžeme počítať s tým, že väčšina z nich bude mať potrebné skúsenosti a znalosti vo vývoji počítačových aplikácií.

Druhým faktorom je jednoduché prevziatie aplikácie a možnosť jej spustenia bez nutnosti zdĺhavej konfigurácie a nutnosti rozsiahlej integrácie do existujúceho systému vyššie spomenutých užívateľov.

Tretím faktorom je možnosť rozšírenia danej aplikácie, tak ako je to v prípade privátnej nadstavby firmy Lundegaard na správu svojej platformy.

Ďalším faktorom je umožnenie iným stranám publikovať rozšírenia, navrhovať zmeny a úpravy v implementácii tejto aplikácie tak, ako to býva zvykom pre open source software.

6.1 Publikácia GitHub

Open source aplikácia bola publikovaná a je verejne dostupná na službe Git. K dispozícií sú všetky zdrojové kódy open source časti tejto aplikácie. Súčasťou tejto publikácie je taktiež návod na okamžité prevziatie a spustenie tejto aplikácie. Ďalej na základe vzoru v tejto práci je možné danú aplikáciu rozšíriť aj pre ďalších užívateľov alebo firmy.

Aplikácia ako taká nie je pripojená k žiadnemu aktívnemu streamu nad Apache Kafka, preto neobsahuje žiadne dáta. V prílohe tejto práce sa nachádza ukážková trieda obsahujúca vzorové dáta prevzaté z

dokumentácie Kafka Connect API. Je možné ňou nahradiť pôvodnú triedu a dostať tak webové rozhranie naplnené týmito dátami.

https://github.com/MartinKocisky/kafka-admin

7 Záver

Cieľom tejto diplomovej práce bolo rozdeliť administratívne rozhranie aplikácie Stream Data Platform Management na open source časť spravujúcu operácie nad Apache Kafka a na privátne rozšírenie potrebné na správu platformy firmy Lundegaard.

Naimplementovať rozšírenie umožňujúce správu nad službami Kafka Connect, následne publikovať online open source časť tohto projektu, a tým umožniť jeho využitie pre ďalšie strany, ktoré by oň mohli mať záujem.

Teoretický úvod zahŕňa informácie o využitých technológiách, s ktorými bolo nutné pracovať a popis ich využitia v rámci tohto projektu. Uvádza ich význam a popisuje ich súčasti, ktoré bolo nutné zvážiť a upraviť v rámci tejto práce.

Táto práca uviedla analýzu štruktúry pôvodného projektu. Popisuje jeho architektúru, implementáciu a spôsob, ktorým aplikácia funguje ako celok. Uvádza jeho hlavné súčasti, moduly a komponenty, ktoré bolo nutné ďalej rozdeliť.

Následne uvádza postup, ktorý bol zvolený na implementáciu, prekážky, ktoré sa v priebehu projektu vyskytli a ich riešenia. Tiež uvádza ukážky kódu a grafického rozhrania danej aplikácie, ktoré boli pridané.

Pre zhrnutie bolo navrhnuté vhodné rozdelenie splňujúce požiadavky a očakávania firmy Lundegaard. Následne bolo toto rozdelenie naimplementované a publikované v spolupráci s danou firmou. Kód bol zverejnený na Git spolu s inštrukciami, ako túto aplikáciu jednoducho spustiť bez nutnosti rozsiahlej konfigurácie.

Ako ďalšie rozšírenie tejto aplikácie navrhujem zjednotenie Docker kontajnerov, z ktorých je táto aplikácia zostavená a ich publikáciu ako jeden spustiteľný Docker obraz na DockerHub.

Všetky požiadavky boli splnené. Rozhranie bolo rozdelené a jeho časti boli publikované. Privátne rozhranie pre firmu Lundegaard bolo naimplementované ako rozšírenie nad open source aplikáciou. Táto časť publikovaná nebola. Aplikácia bola rozšírená o novú funkcionalitu a teraz umožňuje rozsiahlejšiu správu platformy pre firmu Lundegaard.

Bibliografia

1. *Kafka 2.1 Documentation, Kafka Connect* [online]. Apache Software Foundation, 2019 [cit. 2019-03-12]. Dostupné z: https://kafka.apache.org/documentation/#connect.
2. *Kafka Summit* [online]. Apache Software Foundation, 2018 [cit. 2019-03-11]. Dostupné z: https://kafka-summit.org/sessions/kafka-service-offering-tale-security-multi-tenancy/.
3. *How Apache Kafka Inspired Our Platform Events Architecture* [online]. Salesforce Engineering, 2018 [cit. 2019-03-11]. Dostupné z: https://engineering.salesforce.com/how-apache-kafka-inspired-our-platform-events-architecture-2f351fe4cf63.
4. KRCHO, Jozef. *Uživatelské administrativní rozhraní pro fast/stream data platformu*. 2018. Diplomová práca. Masaryk University.
5. *Welcome to Apache ZooKeeper* [online]. Apache Software Foundation, 2019 [cit. 2019-04-28]. Dostupné z: https://zookeeper.apache.org/.
6. *Spring Boot* [online]. Pivotal Software, Inc., 2019 [cit. 2019-03-11]. Dostupné z: https://spring.io/projects/spring-boot.
7. *Spring Boot* [online]. Pivotal Software, Inc., 2019 [cit. 2019-03-11]. Dostupné z: https://docs.spring.io/spring/docs/current/spring-framework-reference/web.html.
8. *Getting Started with Redux* [online]. Dan Abramov a the Redux documentation authors, 2019 [cit. 2019-04-20]. Dostupné z: https://redux.js.org/introduction/getting-started.
9. *React Router Introduction* [online]. React Training 2016-2018, 2019 [cit. 2019-04-26]. Dostupné z: https://github.com/ReactTraining/react-router/blob/v3/docs/Introduction.md.
10. *Dynamic Routing* [online]. React Training 2016-2018, 2019 [cit. 2019-04-22]. Dostupné z: https://reacttraining.com/react-router/core/guides/philosophy.
11. *Apache Kafka* [online]. Brno: Apache Software Foundation, 2019 [cit. 2019-03-11]. Dostupné z: https://kafka.apache.org/intro.

12. *Kafka Monitoring* [online]. Lenses.io Ltd., 2019 [cit. 2019-04-28]. Dostupné z: https://docs.lenses.io/using-lenses/monitoring/.

13. *Kafka Tool* [online]. DB Solo, LLC., 2019 [cit. 2019-04-28]. Dostupné z: http://www.kafkatool.com/.

14. *Kafka Manager* [online]. Yahoo Inc., 2019 [cit. 2019-04-28]. Dostupné z: https://github.com/yahoo/kafka-manager.