SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SPARK INTEGRÁCIA S TECHNOLÓGIAMI ELASTICSEARCH A KUBERNETES

SEMINÁRNA PRÁCA

2022 Bc. Juraj Lapčák, Bc.Martin Kubečka, Bc.Juraj Budai, Bc. Ľubomír Ševčík, Bc.Viliam Alakša

SLOVENSKÁ TECHNICKÁ UNIVERZITA V BRATISLAVE
FAKULTA ELEKTROTECHNIKY A INFORMATIKY

SPARK INTEGRÁCIA S TECHNOLÓGIAMI ELASTICSEARCH A KUBERNETES

SEMINÁRNA PRÁCA

Študijný program:	Aplikovaná informatika
Predmet:	I-ASOS – Architektúra softvérových systémov
Prednášajúci:	RNDr. Igor Kossaczký, CSc.
Cvičiaci:	Ing. Stanislav Marochok

Bratislava 2022 Bc. Juraj Lapčák, Bc.Martin Kubečka, Bc.Juraj Budai, Bc. Ľubomír Ševčík, Bc.Viliam Alakša

Obsah

Zoznam obrázkov a tabuliek

1 Apache Spark

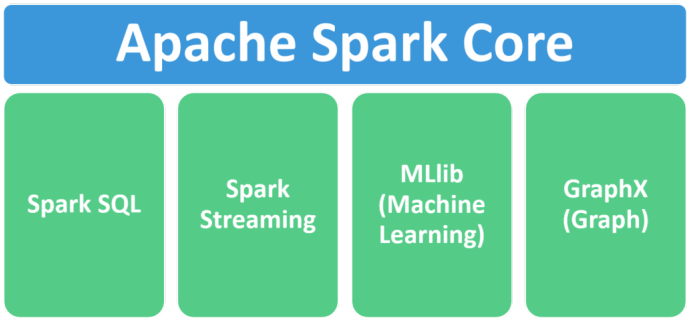
Apache Spark je multi-jazyčný distribuovaný nástroj pre prácu s veľkým objemom dát. Spark vznikol v roku 2009 na UC Berkeley's. V roku 2013 prešiel pod Apache licenciu a od tej doby sa stal jedným z najaktívnejších Apache projektov. Apache Spark je dostupný pre mnoho programovacích jazykov, ako napríklad Java, Scala, Python, R a ďalšie. V rámci implementácie a demonštrácie nášho zadanie sme používali Apache Spark API pre jazyk Java.

Najväčšou výhodou Sparku je jeho horizontálna škálovateľnosť a možnosť pracovať v in-memory režime, čo v praxi znamená, že jednotlivé výsledky mapreduce operácií nebudú ukladané na disk, ale priamo do operačnej pamäte serverov, čím dokáže dosahovať až 100 násobné zvýšenie výkonu oproti klasickej implementácií MapReduce. Túto výhodu využívajú najmä iteratívne algoritmy z prostredia strojového učenia, pri ktorej sa tie isté dáta spracúvajú niekoľkokrát. No aj pri použití klasických diskov sa dosahuje 10 násobné zvýšenie rýchlosti výpočtu. Spark obdržal aj rekord v zatriedení 100TB dát v čase 23 minút za použitia 206 serverov bez použitia in-memory technológie

1.1 Komponenty Apache Spark

Jadro Sparku je tvorené štyrmi základnými knižnicami [1]:

- Spark SQL
- Spark Streaming
- Spark MLlib
- GraphX



Obr. 1: komponenty Apache Spark

Spark SQL je knižnica na prácu so štrukturovanými dátami. Jej súčasťou sú konektory na rôzne databázy, či už z relačného sveta alebo sveta NoSQL databáz. Spark SQL prináša do Sparku možnosť písania dotazov nad dátami v SQL jazyku. Vo výsledku sa SQL dotazy prepíšu do map-reduce programov. O ich výpočet sa už postará Spark. Tu je možné kombinovať a prepájať rôzne zdrojové systémy. Môžeme teda spojiť dve tabuľky, pričom každá bude pochádzať z úplne iného systému.

Spark Streaming je knižnica, ktorá dovoľuje spracovať a analyzovať dáta v reálnom čase. Vstupné dáta, ktoré vchádzajú na spracovanie do Spark Streamingu sa rozdelia na snímky zvané DStreami. Každý snímok obsahuje dáta za určité časové obdobie. Nad DStreamami môžeme následne robiť analýzy a agregácie, pričom je možné využiť akcie a transformácie popísané. Jednotlivé výsledky môžeme ukladať na rôzne úložiská, alebo ich ďalej spracovávať.

MLlib je knižnica, ktorá združuje distribuované algoritmy pre strojové učenie a štatistiku. Vzhľadom na to, že väčšina algoritmov pre strojové učenie je založená na princípe iteratívneho prechádzania tej istej dátovej štruktúry, môžu tieto algoritmy ťažiť zo schopnosti Sparku uloženia všetkých dát do operačnej pamäte serverov. Knižnica obsahuje implementácie mnohých algoritmy pre účely klasifikácie, regresie a klastrovania. Má zakomponované algoritmy pre delenie dát na trénovacie a testovacie množiny, ako aj implementáciu cross-validácie.

V neposlednom rade, GraphX je knižnica obsahujúca množstvo distribuovaných algoritmov pre prácu s grafmi.

2 Elasticsearch

Elasticsearch [2] je distribuovaný vyhľadávací a analytický nástroj pre všetky typy údajov. Je postavený na Apache Lucene. Po prvýkrát bol vydaný v roku 2010 spoločnosťou ElasticSearch N.V.

2.1 Základné pojmy

Index je zbierka dokumentov, ktorá sa používa na ukladanie a čítanie dát. Index je definovaný jedinečným názvom indexu respektive kľúča. Používa sa na vyhľadávanie, skenovanie alebo mazanie dát z dokumentov. Invertovaný index môžeme prirovnať k hľadaniu stránky knihy, ktorá obsahuje určité slovo. Jeden index môže obsahovať jeden typ údajov s vlastnou dátovou štruktúrou.

Shard je podmnožina dokumentov indexu. Elasticsearch používa "shards", keď je prekročený limit úložiska. Umožní tak index rozdeliť na menšie časti ktore sa daju ďalej rozdeliť medzi viacero serverov.

Dokument je hlavnou a základnou jednotkou informácií v Elasticsearch a je reprezentovaný vo formáte JSON. Tieto dokumenty je možné ukladať a indexovať. Takýto index má jeden alebo viac dokumentov a dokument má jeden alebo viac polí.

Mapovanie je definícia schémy pre index.

Node (uzol) je inštancia procesu Elasticsearch. Ide o server, ktorý ukladá údaje a je súčasťou indexových a vyhľadávacích funkcií Klastra.

Klaster sa skladá z jedného alebo viacerých uzlov (serverov), ktoré ukladajú všetky údaje a poskytujú možnosti indexovania a vyhľadávania vo všetkých uzloch. Každý klaster má jeden aktívny hlavný uzol, ktorý sa volí automaticky (napr. pri zlyhaní aktuálneho hlavného uzla).

Replika je mechanizmus, ktorý Elasticsearch používa na riešenie zlyhaní bez straty údajov, ako je napríklad vypnutie uzla. Je to kópia primárneho shardu a môže sa používať na vyhľadávanie rovnako ako pôvodný shard.

2.2 Využitie Elasticsearch

Elasticsearch je schopný indexovat mnoho typov údajov. Môže byť teda využitý vo viacerých odvetviach, ako napríklad vyhľadávanie v aplikáciách, vyhľadávanie vo webových stránkach, logovanie a analýza logov, monitorovanie kontajnerov, monitorovanie výkonu aplikácií, analýza a vizualizácia geopriestorových údajov, bezpečnostné monitorovanie a mnoho ďalších.

2.3 Teoretický princíp fungovania Elasticsearch

Elasticsearch funguje na princípe vyhľadávania, pričom spravuje dokumentovo orientované a pološtruktúrované údaje. Funguje na architekture "zdielania ničoho", čo predstavuje architektúru, v ktorej je každá požiadavka na aktualizáciu obslúžená jedným uzlom.

Uzly tak pristupujú nezavisle na sebe, k tej istej pamäti alebo úložisku. Primárnou dátovou štruktúrou ktorú Elasticsearch používa, je invertovaný index spracovavaný pomocou API Apache Lucene. Invertovaný index je mapovanie každého jedinečného slova (kľúča) na zoznam dokumentov obsahujúci toto slovo. Tento princíp umožňuje veľmi rýchlo vyhľadávať dokumenty, v ktorých sa nachádzajú jednotlive slova (kľúče).

Informacie o indexe sa ďalej ukladajú do partícií nazývaných shards. Elasticsearch dokáže dynamicky distribuovať a prideľovať shardy uzlom v klastri alebo ich replikovať. Vďaka tomuto mechanizmu je flexibilný, pokiaľ ide o distribúciu údajov. Redundanciu možno zabezpečiť rozmiestnením replík shardov (kópie primárnych shardov) do rôznych uzlov klastra.

3 Spôsoby integrácie Elasticsearch

V tejto časti si priblížime naše postupy pri integrácii Elasticsearch s 2 rôznymi spôsobmi, a to pomocou technológie Docker a prostredníctvom spustiteľného súboru na lokálnom systéme.

3.1 Integrácia pomocou technológie Docker

V prvom kroku je potrebné mať nainštalovaný Docker alebo Docker Desktop pre operačný systém Windows. Následná inštalácia docker kontajnerov pre Elasticsearch [3] a jeho grafické rozhranie Kibana nebola náročná. Postup spočíval v stiahnutí obrazu Elasticsearch pomocou uvedeného príkazu.

```
docker pull docker.elastic.co/elasticsearch/elasticsearch:8.5.2
```

Následne spustíme Docker kontajner príkazom nižšie.

```
docker run --name es-node01 --net elastic -p 9200:9200 -p 9300:9300 \
  -t docker.elastic.co/elasticsearch/elasticsearch:8.5.2
```

Po pustení sa nám v okne príkazového riadku vypíše heslo pre základného používateľa elastic, ktoré sme následne použili pre prihlásenie do služby Kibana. Taktiež sa nám zobrazil enrollment token pre inicializáciu tejto služby. Následne sme stiahli a spustili Kibanu.

```
docker pull docker.elastic.co/kibana/kibana:8.5.2
docker run --name kib-01 --net elastic -p 5601:5601 \
  docker.elastic.co/kibana/kibana:8.5.2
```

Vo webovom prehliadači sme sa prihlásili na adrese http://localhost:5601/ pomocou prihlasovacieho mena elastic a hesla, ktoré sme spomenuli vyššie pri spustení Elasticsearch kontajnera. Po prihlásení sme spustili Enterprise search integráciu. Následne bola potrebná konfigurácia v kóde projektu, kde sme použili Apache Spark na zapisovanie dát do Elasticsearch databázy pomocou nasledovnej konfigurácie, pričom appName je reťazec, ktorý reprezentuje názov našej aplikácie, userName reťazec prihlasovacieho mena do služby (elastic), password teda heslo do služby (vygenerované docker kontajnerom). Pre naše riešenie sme zvolili ako cieľový endpoint localhost s portom 9200.

```
SparkConf config = new SparkConf().setAppName(appName).setMaster("local");
. . .
config.set( "spark.es.net.http.auth.user" , userName)
config.set( "spark.es.net.http.auth.pass" , password)
. . .
config.set( "es.nodes" , "localhost")
config.set( "es.port" , "9200")
config.set( "es.nodes.wan.only" , "false" )
config.set( "es.net.ssl" , "true" )
config.set( "es.net.ssl.keystore.location" , "hdfs:///user/uappdaml/ca.p12")
. . .
```

V poslednom uvedenom riadku sme nastavili autorizáciu pomocou TLS a cieľ úložíska šifrovacieo kľúču k súbpru ca.p12. Kľúč sme mohli získať z docker kontajneru es01 pomocou spustenia nástroja keytool a uložiť ho do súboru ca.p12 nasledujúcim príkazom.

```
docker exec es01 keytool -importkeystore \
  -srckeystore ./config/elastic-certificates.p12 \
  -srcstorepass "" -srcalias ca \
  -destkeystore ./ca.p12
```

3.2 Lokálne pomocou spustiteľného súboru

Integrácia je možná na operačných systémoch Windows, MacOS a Linux. Ako prvé je potrebný Elasticsearch, minimálne verzia 6.0.0, nakoľko prechádzajúce verzie už nie sú podporované. Pre inštaláciu na operačnom systéme Windows používatelia môžu stiahnuť Elasticsearch ako ZIP archív [4] a následné spustiť elasticsearch-6.0.0/bin/elasticseach.

Používatelia operačných systémov Linux a MacOS môžu archív stiahnuť pomocou nástroja wget uvedeným príkazom.

```
wget \
https://artifacts.elastic.co/downloads/elasticsearch/elasticsearch-6.0.0.tar.gz
```

Následne stiahnutý archív rozbalíme a pomocou terminálu spustíme súbor.

```
./elasticsearch-6.0.0/bin/elasticsearch
```

Týmto sme spustili lokálne Elasticsearch na porte 9200. Správnu inštaláciu môžeme overiť pomocou HTTP dopytu.

```
curl http://localhost:9200
```

Pri správnej inštalácii a nasadení Elasticseach vidíme nasledujúcu odpoveď.

```
{
  "name" : "gIIS8X9",
  "cluster_name" : "elasticsearch",
  "cluster_uuid" : "hfwabqLsR2iJ0uezLyl4tQ",
  "version" : {
    "number" : "6.0.0",
    "build_hash" : "8f0685b",
    "build_date" : "2017-11-10T18:41:22.859Z",
    "build_snapshot" : false,
    "lucene_version" : "7.0.1",
    "minimum_wire_compatibility_version" : "5.6.0",
    "minimum_index_compatibility_version" : "5.0.0"
  },
  "tagline" : "You Know, for Search"
}
```

Práca a konfigurácia pomocou lokálneho spustitelného súboru bola jednoduchšia a odporúcame ju každému, kto sa chce naučiť prácu s Apache Spark v kombinácii s Elasticsearch. Na druhej strane, tento postup nie je vhodný na prácu v produkčnom prostredí, kde odporúčame nasedenie docker kontajneru na server, poprípade AWS Cloud Lambdu alebo Kubernetes. Jednoduchý spôsob nasadzovania produkčnej verzie spomínanej kombinácie Apache Spark a Elasticsearch je vytvorenie docker-compose YAML súboru a následné spustenie pomocou príkazu docker-compose up s daným súborom.

Príklad docker compose YAML súborov pre viac uzlov Elasticsearch môžeme prevziať z oficiálnej dokumentácie alebo na fóre.

4 Práca Apache Spark v kombinácii s Elasticsearch

Vytvorili sme ukážkový projekt v jazyku Java, v ktorom sme načítali CSV súbor so záznamami top hitov v rámci hudobnej streamovacej služby Spotify [5]. Pre možnosť práce s Apache Spark sme použili Maven repozitár, ktorý nám umožnil automatizovanú prácu s Apache Spark. Keďže pracujeme v Jave bola potrebná verzia 2.1.1 Sparku, ten sme importovali pomocou nasledujúcich dependencii v pom.xml súboru v našom projekte.

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-core_2.11</artifactId>
  <version>2.4.8</version>
</dependency>
```

Pre vytvorenie základných Apache Spark funkcionalít. Následne:

```
<dependency>
  <groupId>org.apache.spark</groupId>
  <artifactId>spark-sql_2.11</artifactId>
  <version>2.4.8</version>
  <scope>provided</scope>
</dependency>
```

Pre umožnenie práce so Spark datasetmi a načítanie z CSV súboru. Pre možnosť uloženia nami načítaného súboru do Elasticsearch databázy bolo potrebné pridať aj dependenciu elasticsearch s minimálnou verziou 6.0.0. pretože skoršie verzie sú už aktuálne deprecated.

```
<dependency>
  <groupId>org.elasticsearch</groupId>
  <artifactId>elasticsearch-spark-20_2.11</artifactId>
  <version>6.0.0</version>
</dependency>
```

Implementovali sme triedu CsvReader, ktorá sa stará o konfiguráciu Spark session, tak aby bol prístupná už predom pustená služba Elasticsearch na lokálnej mašine, na porte 9200.

```
this.spark = SparkSession.builder().appName(appName).master("local")
                .config("es.index.auto.create", "true")
                .config("es.nodes", "localhost")
                .config("es.port", "9200")
                .getOrCreate();
```

Ktorá má dve verejné metody readCsv a close. readCsv metóda má jeden vstupný parameter reťazec csvFile. Ktorá pomocou vytvorenej inštancie Spark session prečíta Dataset z cieľovej cesty, ktorú zadal používateľ pomocou csvFile reťazca a následne pomocou statického objektu JavaEsSparkSQL importovaného z elasticsearch spark balíčka uloží do typu "spotify/records" na localhost:9200 inštancii Elasticsearch.

```
Dataset<Row> csv = spark.read().format("csv")
  .option("header", "true").load(csvFile);
csv.show();
JavaEsSparkSQL.saveToEs(csv.limit(100000), "spotify/records");
```

Po uložení datasetu na localhost môžeme vyhľadávať spotify dáta vložené do Elasticsearch pomocou volania "http://localhost:9200/spotify/records/_search". A vidíme odpoveď, ktorá je vo formáte JSON.

Z odpovede sme vyčítali správny počet záznamov a to sto tisíc a všetky záznamy so správnymi vlastnosťami. Počas vykonávania ukladania do Elasticsearch sme si mohli všimnúť, že dáta sú posielané po častiach .

Následne pomocou volania na rovnakú adresu spomenutú vyššie s request telom:

```
{
  "query": {
    "match": {
  "region": "Mexico"
    }
  }
}
```

sme získali odpoveď so záznamami pesničiek, ktoré boli najlepšie za posledné roky vrámci regiónu Mexiko a tých je 1803.

Podobným spôsobom sme vytvorili triedu EsReader, ktorá využíva Elasticsearch už s dátami na indexe spotify/records. Konštruktor tiredy vytvorí novú Spark konfiguráciu s už spomínaným endpointom a vytvorí nový JavaSparkContext s danou konfiguráciou.

```
public EsReader() {
    SparkConf config = new SparkConf().setAppName(appName).setMaster("local");
    config.set("es.index.auto.create", "true");
    config.set("es.nodes", "localhost");
    config.set("es.port", "9200");
    this.jsc = new JavaSparkContext(config);
   }
```

Trieda obsahuje metódu, ktorá načíta záznamy z Elasticsearch endpointu z indexu spotify-/records a vypíše prvých 20 záznamov na overenie.

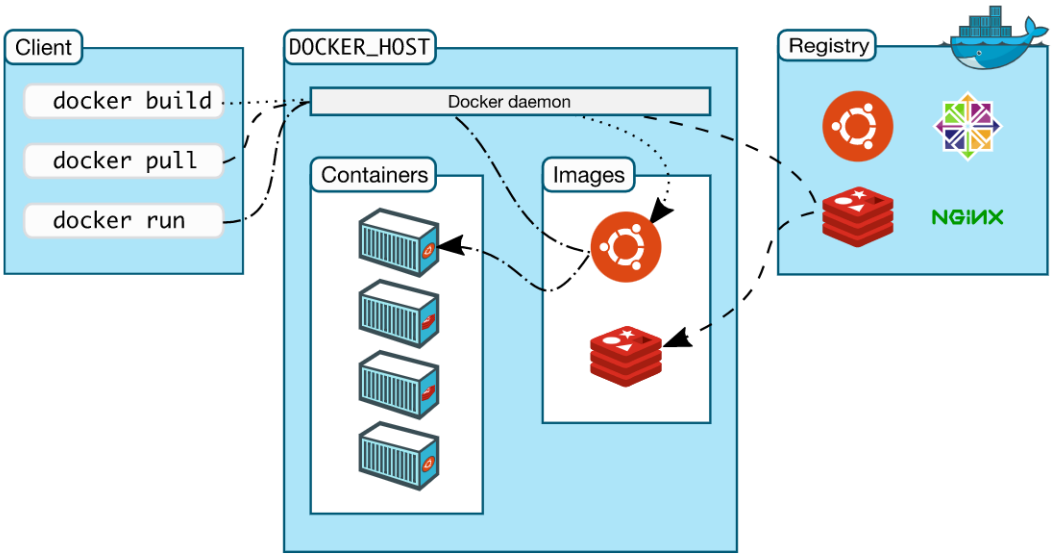
```
public void readEsToCsv() {
       SQLContext sql = new SQLContext(jsc);
       Dataset<Row> df = JavaEsSparkSQL.esDF(sql, "spotify/records");
       System.out.println(df.count());
       df.show();
       this.jsc.close();
   }
```

5 Technológia Docker

Technológia Docker je open-source platforma na vývoj a spúšťanie aplikácií. Docker umožňuje oddeliť aplikáciu od infraštruktúry, aby sa dal software rýchlejšie nasadzovať. Taktiež skracuje čas testovania a oneskorenia medzi vývojom a nasadením kodu. Docker poskytuje možnosť zabaliť aplikáciu do "balíčkov" (z angl. package) a spustiť ju v izolovanom prostredi nazývanom kontajner. Izolácia umožňuje spustiť na jednom operačnom systéme viacero kontajnerov súčasne.

5.1 Architektúra

Docker používa architektúru typu Klient-Server. Klient Docker komunikuje s démonom (z angl. daemon) Docker, ktorý spúšťa a distribuuje jednotlivé kontajnery.



Obr. 2: Docker architektúra

6 Kubernetes

Kubernetes [6] je open-source kontajnerový systém na automatizáciu nasadzovania, škálovania a správy softvéru. Moderný softvér sa čoraz častejšie spúšťa ako "flotily" kontajnerov, niekedy nazývané mikroslužby. Kompletná aplikácia sa môže skladať z mnohých kontajnerov, ktoré musia spolupracovať špecifickými spôsobmi. Kubernetes je softvér, ktorý mení súbor fyzických alebo virtuálnych hostiteľov (serverov) na platformu, ktorá

- hosťuje kontajnerové pracovné záťaže, poskytuje im výpočtové, úložné a sieťové zdroje
- automaticky spravuje veľké množstvo aplikácií v kontajneroch, teda udržiava ich v dobrom stave a dostupné tým, že sa prispôsobuje zmenám a výzvam

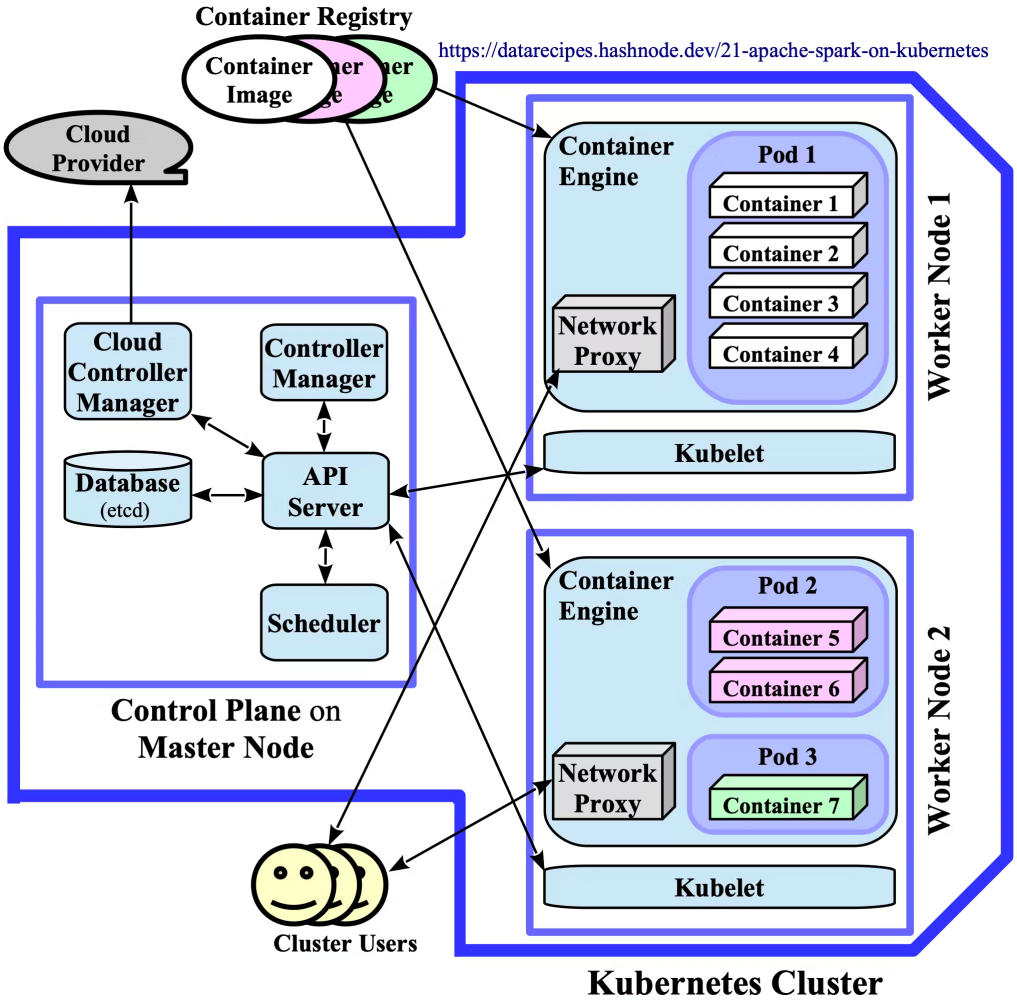
6.1 Použitie

Pomocou Kubernetes môžeme:

- Orchestrovať kontajnery na viacerých hostiteľoch.
- Lepšie využívať hardvér na maximalizáciu zdrojov potrebných na prevádzku vašich podnikových aplikácií.
- Riadiť a automatizovať nasadzovanie a aktualizácie aplikácií.
- Pripájať a pridávať úložisko na spustenie stavových aplikácií.
- Škálovať kontajnerové aplikácie a ich zdroje za chodu.
- Deklaratívne spravovať služby, čo zaručuje, že nasadené aplikácie budú vždy bežať tak, ako ste ich zamýšľali.
- Kontrolovať stav a samo regenerovať aplikácie pomocou automatického umiestňovania, automatického spúšťania, automatickej replikácie a automatického škálovania.

6.2 Návrh a použitie

Nasledujúci diagram vizualizuje vyššie opísané konštrukcie K8s a ich vzťahy v malom klastri, ktorý pozostáva z jedného hlavného a dvoch pracovných uzlov.



Obr. 3: Kubernetes cluster

V našom prípade všetky procesy K8s bežia v kontajneroch Docker na jednom vývojovom počítači. V takýchto lokálnych kontextoch sa tri obdĺžniky, ktoré symbolizujú hlavné a pracovné uzly, zhodujú a zodpovedajú lokálnemu stroju. Napriek tomu sú lokálne nasadenia "plnohodnotnéä chýba len komponent Cloud Controller Manager, pretože v samostatnom kontexte nie je potrebný.

Docker Desktop a minikube sú dva populárne nástroje na vytváranie plnohodnotných, ale lokálnych klastrov Kubernetes. Systémové komponenty týchto klastrov bežia ako

kontajnery Docker na vývojárskom počítači. Minikube je možné nasadiť aj ako virtuálny stroj alebo priamo na systéme. Oba nástroje sú k dispozícii v systémoch macOS a Windows, Linux podporuje len minikube. Ak sme uviedli, pri našom riešení sme si vybrali minikube [7].

6.3 Konfigurácia a nasadenie softvéru

6.3.1 Základné nastavenia

Softvér Kubernetes nainštalujeme v našom prípade pomocou homebrew ale je možnosť ho stiahnuť aj ako spustiteľný binárny súbor pomocou nástroja curl.

Postup inštalácie a overenie konfigurácie kubectl

1. Spustíme inštalačný príkaz: brew install kubectl
2. Otestujeme, či je nainštalovaná verzia aktuálna: kubectl version –client
3. Skontrolujeme, či je kubectl správne nakonfigurovaný, získaním stavu klastra: kubectl cluster-info

V prípade, že sa sa zobrazí odpoveď URL, kubectl je správne nakonfigurovaný na prístup k nášmu klastru. Ak sa zobrazí správa podobná nasledujúcej, kubectl nie je správne nakonfigurovaný alebo sa nedokáže pripojiť ku klastru Kubernetes.

```
The connection to the server <server-name:port> was refused -
did you specify the right host or port?
```

Ak plánujeme spustiť klaster Kubernetes na svojom notebooku, teda lokálne, budeme v prvom rade potrebovať nainštalovať nástroj, ako napríklad Minikube, a potom znovu spustiť vyššie uvedené príkazy. https://minikube.sigs.k8s.io/docs/start/

```
curl -LO \
https://storage.googleapis.com/minikube/releases/latest/minikube-darwin-arm64

sudo install minikube-darwin-arm64 /usr/local/bin/minikube
```

V základe minikube ponúka 2 CPU a 4 GB pamäte. Tieto hodnoty sú nižšie ako oficiálne odporúčané 3 CPU a 4 GB pamäte, aby bolo možné spustiť jednoduchú aplikáciu Spark s jedným vykonávacím programom. Preto sa odporúča spustiť klaster minikube, ktorý dokáže prideliť viac zdrojov, aby sa neskôr mohla splniť požiadavka na dva Spark exekutory. Žiaľ, nie je možné dynamicky zvyšovať zdroje (ako v prípade Docker Desktop), takže musíme spustiť minikube s docker driverom, ktorý takéto možnosti podporuje.

```
minikube start driver=docker --cpus 5 --memory 6000
```

Ak už bol Docker v systéme prítomný, je vhodné skontrolovať kontext Docker minikube pomocou:

```
minikube docker-env
```

Dashboard Kubernetes je populárnym členom ekosystému K8s a je predinštalovaný v minikube, keďže používame tento nástroj dashboard nie je potrebné inštalovať ani vytvárať roly pre prístup. Tento dashboard spustíme príkazom: minikube dashboard

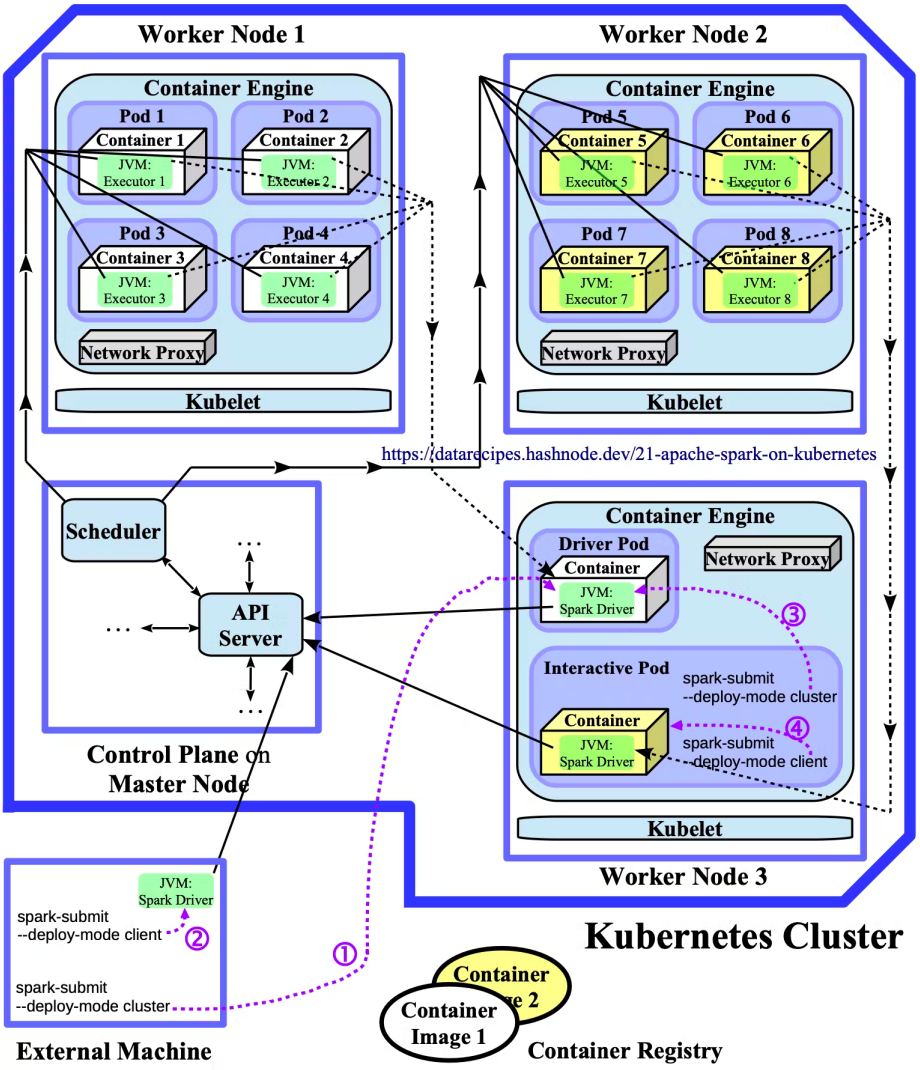
V predvolenom webovom prehliadači by sa mala automaticky otvoriť nová karta s používateľským rozhraním Dashboard. Príkaz spustí dva ďalšie pody v novom mennom priestore kubernetes-dashboard. Jeden pod je scraper metrík, druhý obsluhuje používateľské rozhranie.

V tomto kroku môžeme použiť kubernetes na prístup k svojmu novému klastru pomocou nasledujúceho príkazu.

```
kubectl get po -A
```

6.3.2 Príprava prostredia

So Sparkom 3.1.1 sa stala všeobecne dostupnou natívna podpora vykonávania pre Kubernetes. V predchádzajúcich bodoch je vysvetlené, ako možno nainštalovať minikube a ako spustiť vhodný lokálny klaster Kubernetes. Teraz využijeme backend Kubernetes a uvedieme spôsob ako možno natívne posielať aplikácie Spark. Nasledujúci diagram znázorňuje rôzne možnosti nasadenia.



Obr. 4: Možnosti nasadenia Kubernetes

Z priestorových dôvodov nie je zahrnutá väčšina komponentov riadiacej roviny. Vizualizujeme scenár, v ktorom osem kontajnerových vykonávacích jednotiek Spark beží na klastri K8s.

Každý Spark executor beží v jednom kontajnerovom pode. V samostatnom prístupe jeden pod/kontajner často obaľuje viacero exekútorov.

Rôzne aplikácie môžu používať rôzne kontajnerové obrazy. Nie je žiadny medzičlánok Spark Master, aplikácie preto môžu priamo využívať objekty a funkcie Kubernetes, ako sú menné priestory, účty služieb, elasticita a ďalšie.

V prostredí minikube, ktoré sa používame, sa všetky obdĺžniky zhodujú do jedného

obdĺžnika dev machine a všetky procesy bežia v kontajneroch (kontajneroch vnútri) Docker. Na jedno odoslanie aplikácie sa spustia len dva exekútori.

Jedným z parametrov, ktorý zostáva konštantný v uvedených alternatívach ëxterného"nasadenia, je reťazec URL Spark master, ktorý sa odovzdáva príkazom spark-submit. Ovládač Spark sa pripája k serveru API, ktorého koncový bod je na mojom dev počítači podľa správ vypísaných pomocou:

```
kubectl cluster-info
```

```
Kubernetes control plane is running at https://127.0.0.1:50202
CoreDNS is running at
127.0.0.1:50202/api/v1/namespaces/kube-system/services/kube-dns:dns/proxy
```

Server API je súčasťou riadiacej roviny, takže príslušná adresa URL je uvedená na konci položky "Kubernetes control plane is running at". Keď kubectl ukazuje na reálny klaster K8s, príslušný riadok bude pravdepodobne začínať "Kubernetes master is running at http...". V skutočných príkazoch spark-submit musí byť koncový bod prefixovaný k8s://, čo v našom prípade vedie k reťazcu master k8s://https://127.0.0.1:50202. Vynechanie protokolu HTTP (t. j. k8s://127.0.0.1:57844) by bolo tiež prijateľné, pretože v predvolenom nastavení je to https.

Namiesto ručného získavania adresy servera API a jej kopírovania a vkladania do slotu –master v príkazoch odoslania je možné deklarovať vyhradenú premennú prostredia a neskôr sa na ňu odkazovať.

```
K8_API_URL=$(kubectl config view \
  --minify -o jsonpath="{.clusters[0].cluster.server}")
```

Teraz potrebujeme Docker image sparku, ktorý si stiahneme pomocou príkazu.

```
docker pull bdrecipes/spark-on-docker:latest
```

Niektoré predvolené nastavenia v prostredí minikube sú reštriktívnejšie, takže vo všetkých nižšie uvedených podaniach aplikácií je potrebné výslovne uviesť účet služby s príslušnými oprávneniami. Vhodný účet ServiceAccount s názvom bigdata-sa s dostatočnými právami na čítanie/zápis možno vytvoriť pomocou nasledujúcej dvojice príkazov.

```
kubectl create serviceaccount bigdata-sa
kubectl create rolebinding bigdata-rb \
  --clusterrole=edit --serviceaccount=default:bigdata-sa
```

Toto konto ServiceAccount budú používať všetky pody Spark, ktoré budú spustené v priebehu tohto príspevku. Pre zjednodušenie budú všetky pody spustené do predvoleného menného priestoru. Teraz je všetko pripravené na spustenie prvej orchestrovanej aplikácie Spark. Spočiatku bolo možné na backendoch K8s používať len režim nasadenia klastra, podpora klientskeho režimu bola implementovaná vo verzii Spark 2.4. Režim nasadenia ovládača Spark (klaster verzus klient) je ortogonálny k prostrediu spustenia (v rámci klastra verzus mimo klastra), čo vedie k štyrom možným kombináciám. Tieto sú znázornené ako (1) až (4) v diagrame v hornej časti tejto stránky. Zápisníky Jupyter a Spark REPL sú dva populárne príklady aplikácií v klientskom režime.

6.3.3 External cluster deploy

Jedna zo stratégií nasadenia spočíva v spustení procesu odosielania z "vonkajšej"strany klastra K8s, priamo z počítača dev. Pri použití minikube nie je klaster K8s skutočne vzdialený, ale beží v rámci kontajnerov Docker (ak nie sú zvolené iné ovládače), takže prostredie spúšťania a klastera sú identické. V nasledujúcej časti je opísaný režim externého nasadenia klastra, alternatíva externého klienta je preskúmaná neskôr.

Predpripravený image bdrecipes obsahuje distribúciu Spark spolu so všetkými požadovanými aplikačnými artefaktmi. Preto bude každý kontajner, ktorý sa inštanciuje z tohto imagu, vybavený knižnicami Spark a skriptom spark-submit. Skript submit sa však nevyvoláva z kontajnera/podkladu pod ním, takže podobná distribúcia Spark musí byť prítomná v lokálnom spúšťacom prostredí. Na mojom vývojovom počítači bol spark-3.1.1-bin-hadoop3.2.tgz rozbalený v domovskom adresári, skript spark-submit sa preto nachádza v adresári /spark-3.1.1-bin-hadoop3.2/bin/. Výsledkom je nasledujúci príkaz na spustenie QueryPlansDocker.scala proti Kubernetes v režime klastra s dvoma exekutormi:

```
~/spark-3.1.1-bin-hadoop3.2/bin/spark-submit \
 --master k8s://$K8_API_URL \
 --deploy-mode cluster \
 --conf spark.kubernetes.container.image=bdrecipes/spark-on-docker:latest \
 --conf spark.executor.instances=2 \
```

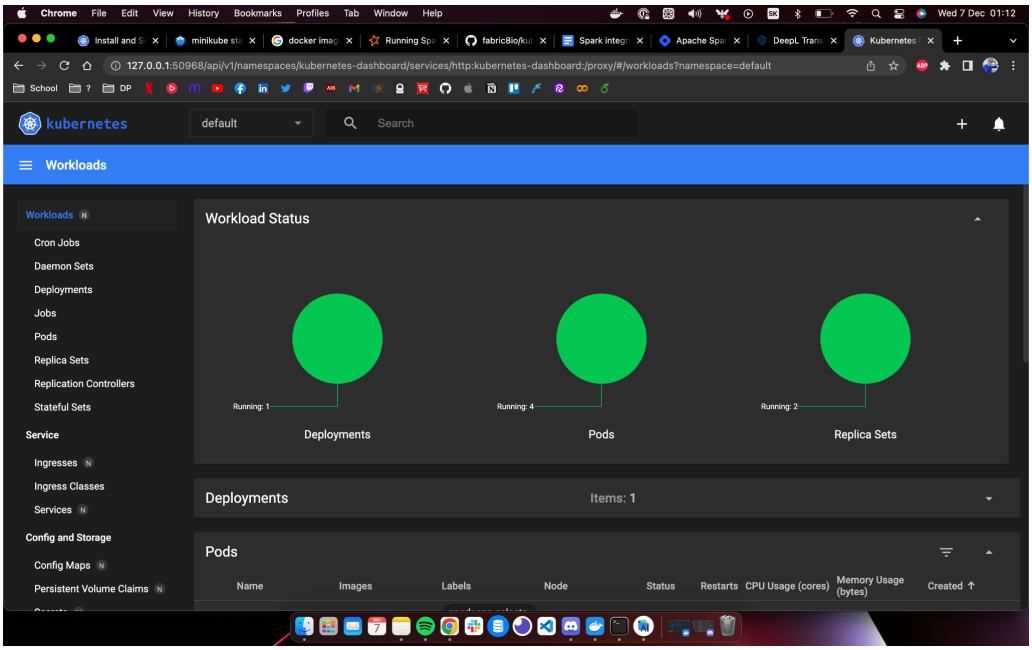
```
--conf spark.kubernetes.driver.pod.name=driver-pod \
--conf spark.kubernetes.authenticate.driver.serviceAccountName=bigdata-sa \
--class module1.scala.QueryPlansDocker \
local:///opt/spark/work-dir/bdrecipes-phil.jar \
/opt/spark/work-dir/resources/warc.sample
```

Dôležitým rozdielom oproti režimu nasadenia klienta, ktorý je opísaný neskôr, je, že posledné dva riadky príkazu (ktoré špecifikujú zdrojový súbor JAR/Python a vstupný súbor) odkazujú na ümiestnenie kontajnera". Vždy, keď sa aplikácie Spark spúšťajú v režime klastra proti backendu K8s, ovládač aj exekutory sú kontajnerizované. Preto sa v lokálnom prostredí, z ktorého sa spúšťa proces odosielania, nemusí nachádzať súbor bdrecipes-phil.jar/query_plans_docker.py ani vstupný súbor programu warc.sample. Počas procesu zostavovania obrazu bdrecipes/spark-on-docker boli všetky relevantné závislosti skopírované do súborového systému obrazu a ich cieľové cesty sú zohľadnené v oboch príkazoch na odoslanie.

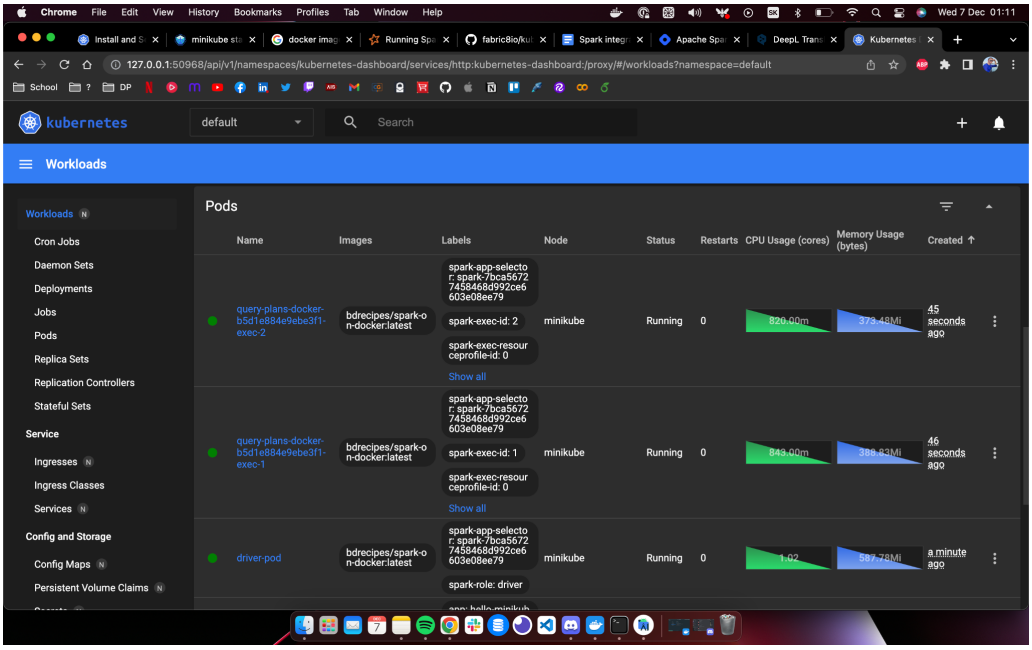
Ovládač Spark sa odpojí od spúšťacieho prostredia a spustí sa v module. Po niekoľkých sekundách by sa mali vytvoriť dva pracovné pody, v ktorých sú vložené dve požadované inštancie vykonávacieho mechanizmu. Zadaním príkazu:

```
kubectl get pods
```

Viackrát za sebou v krátkom časovom slede počas tejto štartovacej fázy sa potvrdí opis. Používateľské rozhranie Dashboard, ktoré bolo povolené v úvode K8s, pekne vizualizuje tri Spark pods.



Obr. 5: Dashboard overview no.1



Obr. 6: Dashboard overview no.2

Keďže sa ovládač Spark odpojí, vidíme len hlavné aktualizácie stavu aplikácie, ako je fáza: V termináli, v ktorom bol vyvolaný proces odosielania, sa zobrazí položka: Running. Logy ovládača poskytujú podrobnejšie informácie. V paneli Dashboard ich možno zobraziť

kliknutím na akciu Logs v pravom stĺpci karty Workloads // Pods. Alebo idiomatickejšie pomocou príkazu.

```
kubectl logs
```

6.3.4 External client deploy

Pred použitím režimu klienta z miestneho prostredia je potrebné vykonať niekoľko úprav rozlíšenia prostriedkov: Najzrejmejšie je, že hodnota odovzdaná do možnosti –deploy-mode sa musí zmeniť z clustera na klienta. Keď sa tento režim používa "zvonku"klastra K8s, ovládač sa neodpojí do podov, ale naďalej beží v relácii terminálu, z ktorej sa volá skript odoslania. Keďže vstupný súbor warc.sample, ktorý spotrebúva QueryPlansDocker.scala, musí byť skenovaný ovládačom Spark a vykonávacími programami, vzniká problém s prístupom k súboru: Vykonávače budú stále kontajnerizované a vstupný súbor je súčasťou ich základného obrazu Docker v adresári /opt/spark/work-dir/resources/warc.sample. Proces ovládača však nebude bežať vo vnútri podov, takže obsah imagu bdrecipes/spark-on-docker je mimo dosahu. Najjednoduchšie riešenie tohto problému s prístupom spočíva v lokálnom vytvorení nového adresára /opt/spark/work-dir/resources/ a umiestnení nekomprimovanej kópie warc.sample do neho.

Poskytovanie zdrojových kódov programu je jednoduchšie, pretože ich nepotrebujú vykonávacie programy Spark, musí k nim pristupovať len proces ovládača: Po kompilácii vetvy KubernetesImage projektu bdrecipes pomocou mvn clean install sa zostavený súbor JAR nachádza na adrese /IdeaProjects/bdrecipes/target/bdrecipes-phil.jar v mojom systéme. Pomocou príkazu možno programy spúšťať z lokálneho spúšťacieho prostredia v klientskom režime.

```
~/spark-3.1.1-bin-hadoop3.2/bin/spark-submit \
 --master k8s://$K8_API_URL \
 --deploy-mode client \
 --conf spark.kubernetes.container.image=bdrecipes/spark-on-docker:latest \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=bigdata-sa \
 --conf spark.executor.instances=2 \
 --class module1.scala.QueryPlansDocker \
 ~/IdeaProjects/bdrecipes/target/bdrecipes-phil.jar \
 /opt/spark/work-dir/resources/warc.sample
```

Po odoslaní aplikácie sa spustia len dva pody, čo je o jeden menej v porovnaní s vyššie uvedeným režimom nasadenia klastra. Ovládač Spark sa neodpája od relácie lokálneho terminálu, takže pod ovládača nie je potrebný. Okrem toho sa protokoly ovládača vypisujú priamo do terminálu, čo je podobné scenáru distribuovaného vykonávania.

6.3.5 Internal cluster deploy

V predchádzajúcich príkladoch bol príkaz spark-submit volaný z "vonkajšej"strany klastra K8s. Spustenie programu môže byť iniciované aj zvnútra klastra, napríklad prostredníctvom sprostredkujúceho interaktívneho/bazového podprogramu. Takéto interaktívne kontajnery možno spustiť pomocou špecifikácie interactive_pod.yaml.

```
kubectl apply -f https://raw.githubusercontent.com/g1thubhub/bdrecipes
 /KubernetesImage/scripts/interactive_pod.yaml pod/interactive-pod created

kubectl exec -it interactive-pod
```

Spúšťanie programu z podov je podobné scenáru aplikácie s viacerými kontajnermi. Podkladový obraz bdrecipes/spark-on-docker obsahuje všetky potrebné artefakty, takže nevzniknú žiadne problémy, keď sa príkazy budú odvolávať len na interné súbory kontajnera. Okrem toho sa v kontajneri Docker nachádza aj samotný skript submit (na adrese /opt/spark/bin/spark-submit), takže všetko sa bude spúšťať z inštancií imagu. Výsledkom je nasledujúca "sebestačná"konfigurácia na spustenie súboru QueryPlansDocker.scala z interaktívneho podkonta v režime klastra.

```
/opt/spark/bin/spark-submit \
 --master k8s://https://kubernetes.default:443 \
 --deploy-mode cluster \
 --conf spark.kubernetes.authenticate.submission.caCertFile=$CACERT \
 --conf spark.kubernetes.authenticate.submission.oauthTokenFile=$TOKEN \
 --conf spark.kubernetes.container.image=bdrecipes/spark-on-docker:latest \
 --conf spark.executor.instances=2 \
 --conf spark.kubernetes.driver.pod.name=driver-pod \
 --conf spark.kubernetes.authenticate.driver.serviceAccountName=bigdata-sa \
 --class module1.scala.QueryPlansDocker \
 local:///opt/spark/work-dir/bdrecipes-phil.jar \
 /opt/spark/work-dir/resources/warc.sample
```

Celkovo sú teraz zapojené štyri pody: Kvôli režimu nasadenia v klastri sa ovládač odpojí od interaktívneho modulu a beží vo vyhradenom module ovládača. Tak ako vo všetkých ostatných scenároch nasadenia, skutočnú výpočtovú prácu vykonávajú dva pody exekútorov.

6.3.6 Internal cluster deploy

Pri tomto bode postupujeme kombináciou 2 a 3 bodu, ale dochádza k neuspešnému pokusu. Za tento neúspešný pokus sú zodpovedné architektúry Spark a Kubernetes podľa dokumentácie [8].

7 Prečo použiť niečo iné ako Kubernetes?

Kubernetes môže byť nadbytočný pre aplikácie, ktoré nepotrebujú veľké škálovanie alebo distribuovaný dizajn [9], napriek tomu, že je ide o veľmi výkonnú a vhodnú technológiu pre rozsiahle nasadenia.

Táto technológia je pomerne zložitá na naučenie aj pre vývojárov a DevOps špecialistov, nehovoriac o menej technických profesiách.

Ďalej uvádzame komplikovanú migráciu na Kubernetes pre existujúce aplikácie. V mnohých prípadoch sa musia refraktorovať komponenty prípadne aj celá architektúra. Aplikácie musia využívať cloudové natívne princípy.

Napriek tomu, že je Kubernetes open-source projekt, tak má veľa skrytých nákladov, teda vysoké celkové náklady na vlastníctvo. Okrem toho, že nasadenie a spustenie je drahé, Kubernetes potrebuje na svoje fungovanie rozsiahlu výpočtovú infraštruktúru. Kubernetes

je mimoriadne výkonný, no ťažko sa učí a ovláda. To viedlo k vytvoreniu platforiem, ktorých cieľom je uľahčiť nasadenie a používanie Kubernetes.

7.1 Container as a Service (CaaS)

Tieto služby nám umožňujú spravovať a spúšťať mnohé kontajnery bez zložitých možností orchestrácie, ktoré poskytuje Kubernetes.

AWS Fargate

Transparentne sa škáluje a zaplatíte podľa výpočtových a pamäťových zdrojov, ktoré

využívate. Je vhodný pre samostatné služby a jednoduché mikroslužby.

Azure Container Instances

Táto služba podporuje Linux aj Windows kontajnery. Microsoft automaticky konfiguruje a

škáluje základné výpočtové zdroje.

Google Cloud Run

Plne spravovaná platforma, ktorá umožňuje spúšťanie Docker kontajnerov ako bezstavové,

automaticky škálované HTTP služby.

7.2 Spravované Kubernetes služby

Tieto služby nám umožňujú spúšťať spravované hosťované Kubernetes „flotily“. Odstraňujú veľkú komplexitu nasadzovania, aktualizovania a spravovania Kubernetes.

Google Kubernetes Engine (GKE)

Google pôvodne vyvinul Kubernetes a stále na ňom aktívne pracuje. Táto služba bola prvá, ktorá slúžila na spúšťanie Kubernetes služieb v cloude. Integruje sa s ďalšími službami Google Cloud a poskytuje riadenie prístupu, zabezpečenie a iné funkcie.

Amazon Elastic Kubernetes Service (EKS)

EKS Beží na najnovšej verzii Kubernetes a je kompatibilný s celým ekosystémom nástrojov Kubernetes. Plne spravuje riadenie Kubernetes a podľa potreby škáluje hlavné uzly, čím zabezpečuje vysokú dostupnosť služby.

Azure Kubernetes Service(AKS)

Zabezpečuje spravovanie Kubernetes „flotíl“, čím znižuje zložitosť riadenia a prevádzky. Na rozdiel od porovnateľných služieb na AWS a Google Cloud, AKS účtuje iba pracovné uzly Kubernetes, pričom ponúka hlavné uzly a správu „flotíl“ zadarmo.

7.3 Jednoduchšie kontajnerové orchestrátory

Kubernetes je najpopulárnejší kontajnerový orchestrátor, ale nie je jediný. Ďalej si predstavíme kontajnerové orchestrátory, ktoré sú menej komplexné na použitie a spravovanie ako Kubernetes.

Docker Swarm

Docker Swarm je natívna vlastnosť Dockeru, ktorá umožňuje spájať a plánovane spúšťať Docker Enginy. Po aktivovaní tejto vlastnosti môžete vytvárať a orchestrovať „flotily“. Taktiež ich môžete monitorovať a škálovať.

Nomad

Táto technológia vyvíjaná spoločnosťou HashiCorp, je flexibilný kontajneroý orchestrátor, ktorý umožňuje nasadiť aj kontajnery aj staršie aplikácie rovnakým spôsobom. Zameriava sa na jednoduchosť použitia.

8 Alternatívy Elasticsearch

Elasticsearch bol pôvodne open-source projekt s licenciou Apache V2.0. V roku 2021 zmenili svoju softvérovú licenciu na duálnu licenciu Elastic Licence a Server Side Public License, ktoré nie sú open-source. Niektorí developeri nechceli naďalej používať Elasticsearch a pre to aj vznikol fork tohto projektu s názvom OpenSearch.

8.1 OpenSearch

OpenSearch je distribuovaný, komunitou riadený, licencovaný Apache 2.0, open-source vyhľadávací a analytický balík, ktorý sa používa na širokú škálu prípadov použitia, ako je monitorovanie aplikácií v reálnom čase, analýza protokolov a vyhľadávanie webových stránok. OpenSearch poskytuje vysoko škálovateľný systém na poskytovanie rýchleho prístupu a odozvy na veľké objemy údajov s integrovaným vizualizačným nástrojom OpenSearch Dashboards, ktorý používateľom uľahčuje skúmanie ich údajov. OpenSearch je založený na vyhľadávacej knižnici Apache Lucene a podporuje množstvo vyhľadávacích a analytických funkcií, ako je vyhľadávanie k-najbližších susedov (KNN), SQL, detekcia anomálií, plnotextové vyhľadávanie a ďalšie. Je vytvorený z forku Elasticsearch-u verzie 7.10.2 a dá sa integrovať so Spark-om.

8.2 Solr

Solr je open-source vyhľadávací nástroj postavený na knižnici Apache Lucene Library, ktorá je napisaná v jazyku Java. Pozostáva z HTTP/XML webových API rozhraní.Samotné Solr dotazy sú vo forme dokumentov JSON. Tento nástroj sa zameriava hlavne na vyhľadávanie v texte a súvisiace operácie. Solr podporuje JSON, ale táto funkcia bola pridaná nedávno. Keď bol Solr pôvodne vytvorený, jeho hlavným jazykom bol XML. Možnosť škálovania je možná iba s pomocou SolrCloud a zookeeper.

Zoznam použitej literatúry

1. KUBICA, Michal. Distribuované prostredie. 2016, č. 21/46. Dostupné tiež z: http://www.dcs.fmph.uniba.sk/bakalarky/obhajene/getfile.php/kubica_michal_anomaly_detection.pdf.
2. What is Elasticsearch? [N.d.]. Dostupné tiež z: https://www.elastic.co/what-is/elasticsearch.
3. Start a multi-node cluster with Docker Compose. [N.d.]. Dostupné tiež z: https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html#docker-compose-file.
4. Install Elasticsearch with .zip on Windows. [N.d.]. Dostupné tiež z: https://www.elastic.co/guide/en/elasticsearch/reference/current/zip-windows.html#zip-windows.
5. Spotify Charts. [N.d.]. Dostupné tiež z: https://www.kaggle.com/datasets/dhruvildave/spotify-charts.
6. Kubernetes Documentation. [N.d.]. Dostupné tiež z: https://kubernetes.io/docs/home/.
7. minikube start. [N.d.]. Dostupné tiež z: https://minikube.sigs.k8s.io/docs/start/.
8. Cluster Mode Overview. [N.d.]. Dostupné tiež z: https://spark.apache.org/docs/latest/cluster-overview.html.
9. Kubernetes Alternatives and Why You Need Them. [N.d.]. Dostupné tiež z: https://www.aquasec.com/cloud-native-academy/kubernetes-101/kubernetes-alternatives/.