BILTEGIRATUTAKO DATUEN PROZESATZEA

Manex Sorarrain eta Aner Garatea

AURKIBIDEA

SARRERA

Lan honetan, Docker Compose bidez Kudu+Impala ingurune bat hedatuko dugu eta biltegiratutako datuak Bussiness Intelligence tresna baten bidez bistaratuko ditugu

Kudu abiadura handiko datu-biltegiratze sistema bat da, datu multzo handiekin denbora errealean lan egiteko diseinatua.

Bestalde, Impala errendimendu handiko datuak aztertzeko tresna bat da, Hadoop-en gordetako datuetan SQL kontsulta interaktiboak egiteko aukera ematen duena.

Kudu eta Impala batera erabil daitezke Hadoop-en gordetako datu multzo handietan denbora errealean datuen analisia egiteko. Kudu-k datuen biltegi azkar eta eraginkor gisa jokatzen du, Impala-k, berriz, SQL interfaze interaktibo bat eskaintzen du datuak kontsultatzeko eta aztertzeko. Elkarrekin, Kudu eta Impala-k datu handien analisirako irtenbide osoa eta eskalagarria eskaintzen dute.

Bestalde, Bussiness Inteligence-ren helburu nagusia datuak informazio baliagarri bihurtzea da, enpresek gertakari eta datu errealetan oinarrituta erabakiak har ditzaten. Lan honetan, BI tresna bezala, Superset plataforma erabili dugu.

Datubasea, New York-eko taxiek egindako bidaiak biltzen dituen Parquet datubase bat da., https://www.nyc.gov/site/tlc/about/tlc-trip-record-data.page hain zuzen ere.

.

PROIEKTUAREN GARAPENA

Lanaren oinarrizko azpiegitura honako elementuez osatuta dago:

- Kudu azpiegitura bat, docker bidez hedatutakoa.
- Impala irudi bat, kudura konektatutakoa.
- Superset irudi bat.

Azpiegiturak ongi funtziona dezan hurrengo pausuak jarraitu ditugu:

1. Parquet fitxategia deskargatu eta data karpeta batean gorde.
    a. wget

       https://s3.amazonaws.com/nyc-tlc/trip+data/yellow_tripdata_2022-01.parquet

2. Enuntziatuko git clone aginduarekin errepositorioa jaitsi. Honen barruan **quickstart.yml** fitxategia aldatu dugu, honetan impala kontainerraren konfigurazioa gehitu dugu, eta egitura guztia sare berean egon dadin, sare bat sortu dugu bridge motakoa. Behin hori eginda, sare hau esleitu diogu irudi bakoitzari. Impalaren konfigurazioan data bolumena irudi barruko data karpetara ere mapatu dugu.

3. Kudu eta impala kontainerrak jaurti
    a. **docker compose -f quickstart.yml up -d** komandoa erabiliz

4. Ondoren, Impala kanpo taula bat sortu dugu Parquet datuetara seinalatzen duena, gainera, kudu taula bat ere sortu dugu Impalan aurrekoaren azpi-taula bezala. Hau dena impala-shell ean exekutatu dugu.
    a. **create external table taxi like parquet 'file:///data/taxi.parquet' stored as parquet location 'file:///data/';**

        - "taxi" izeneko Impala-n kanpo taula bat sortzen du, 'file:///data/taxi/taxi.parquet' helbidean dagoen Parquet fitxategiaren egituraren itxura duena. Taulako datuak 'file:///data/taxi/' direktorioan gordeko dira, Parquet formatuan

    b. **create table taxik (pickup timestamp, dropoff timestamp, distance double, primary key(pickup,dropoff)) stored as kudu;**
        - Impalan "taxik" izeneko taula bat sortzen du, hiru zutabe dituena: timestamp motako "pickup" eta "dropoff" eta double motako "distance". Gainera, taulak gako nagusi bat du "pickup" eta "dropoff" zutabeek osatutakoa. Hau, Kudu taula gisa gordeko da.

    c. **insert into taxik select tpep_pickup_datetime, tpep_dropoff_datetime, trip_distance from taxi;**

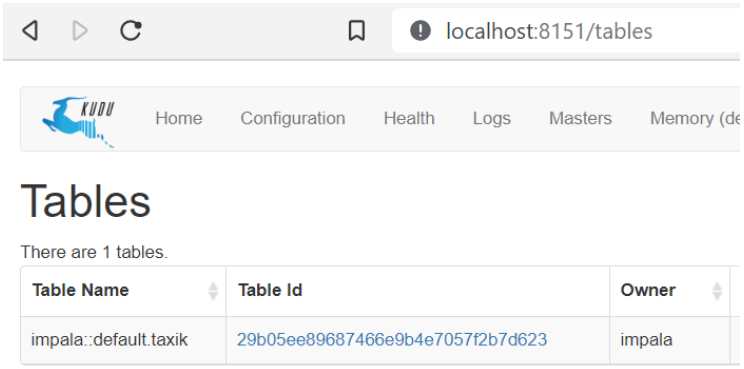
- Azkenik, Parquet taulako datuak Kudu taulan txertatu ditugu goiko komandoa erabiliz.

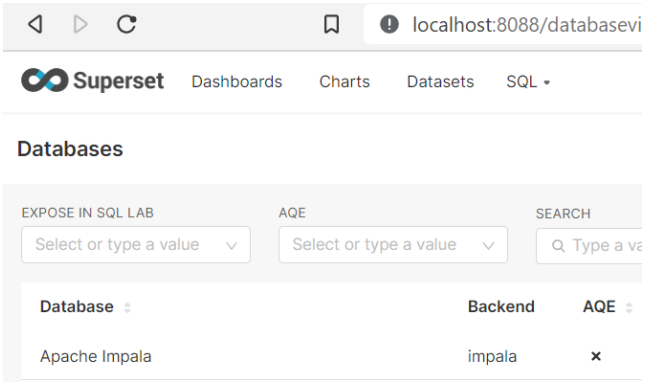
5. Behin Impalara konektatuta, proba-kontsulta bat exekutatu dezakegu honela:

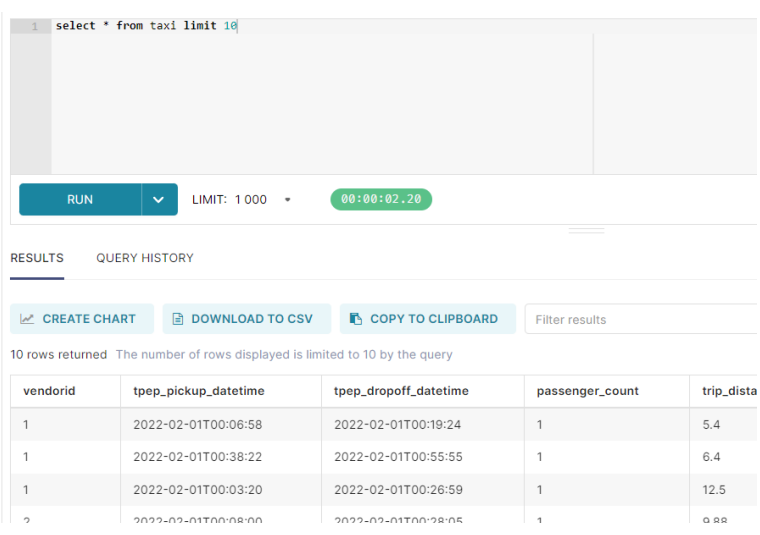
    a. **SELECT COUNT(*) FROM taxik;**
    b. Dena behar bezala konfiguratuta badago, erantzun bat jaso beharko genuke taxik taulako errenkada kopuruarekin.
6. Superseten konfiguraziorako, klaseko tutoriala jarraitu dugu,honako bi salbuespenekin: requirements-local.txt fitxategian impyla, thrift eta thrift_sasl dependentziak jarri ditugu, eta lehen aipatutako sare bera konfiguratu dugu, dena sare berean egon dadin.
7. Ondoren, Superset en datuen jatorria sortu beharko da. Hau egiteko, Superset-en intenfazetik “Origen de datos”en datu base mota bezala Impala jarri beharko dugu eta Datu baserako konexio informazioa sartu.

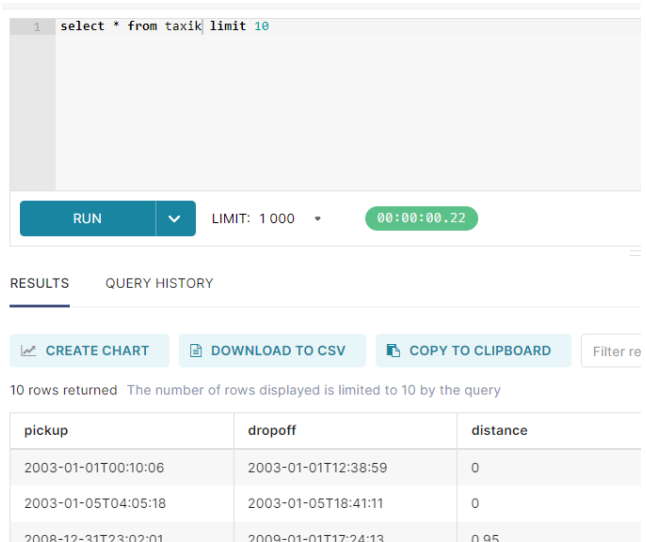
Gure entregagarria jaurtitzeko,
export KUDU_QUICKSTART_IP=$(ifconfig | grep "inet " | grep -Fv 127.0.0.1 | awk '{print $2}' | tail -1) komandoa exekutatu dugu KUDU_QUICKSTART_IP ingurune aldagaia definitzeko. Ondoren, beheko komandoarekin arkitektura guztia martxan jartzen dugu..
‘docker compose -f quickstart -f docker-compose.yml up’

Behin guztia martxan jarrita, kudun taxik taula agertzen da, impala eta superset arteko konexioa ondo dago, impalan ikusten dira superseteko queriak, eta gainera, superseten taxi eta taxik taulen gainean kontsultak egin ditzakegu. Hurrengo argazkietan ditugu hau baieztatzen duten kapturak:







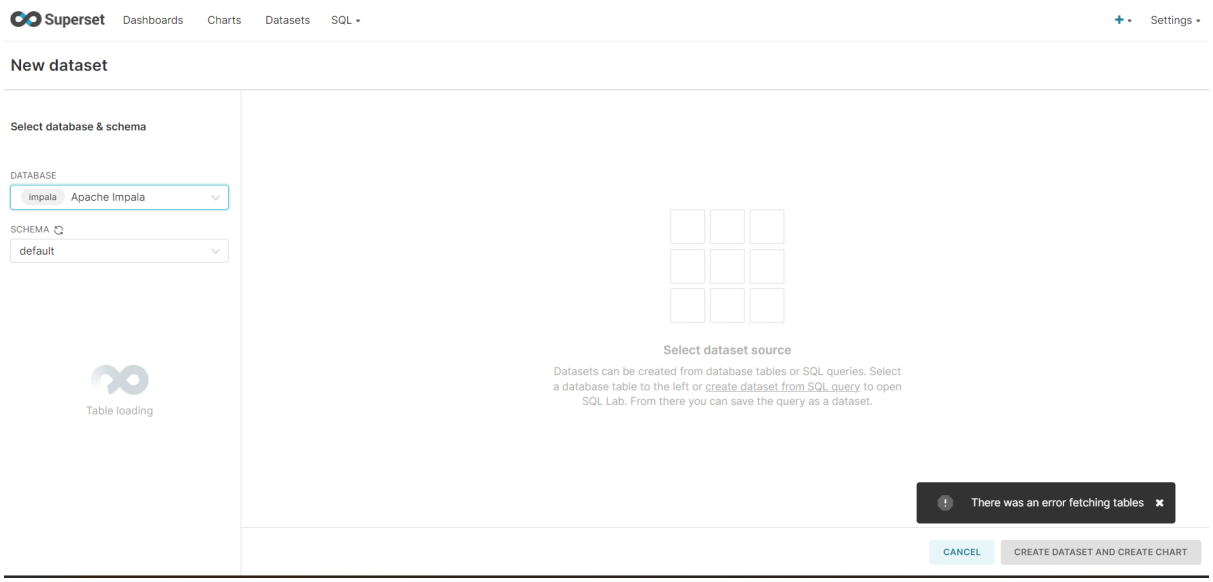


Last 25 Completed Queries

User	Default Db	Statement	Query Type	Start Time	End Time	Duration	Scan Progress	State	# rows fetched	Resource Pool	Details
root	default	`select * from taxik limit 11`	QUERY	2023-05-01 21:28:01.206515000	2023-05-01 21:28:01.271550000	65.035ms	1 / 1 (100%)	FINISHED	11	default-pool	Details
root	default	`select * from taxi limit 11`	QUERY	2023-05-01 21:24:58.111617000	2023-05-01 21:25:00.070570000	1s958ms	1 / 1 (100%)	FINISHED	11	default-pool	Details

Baina grafikak sortzerakoan, honako arazoa sortu zaigu: Grafikak sortzeko taxik taulatik abiatuta dataset bat sortu behar genuen, baina hau sortzerakoan, honako mezua lortu eta ezinezko egiten zen hau sortzea.

Hemen dataseta sortu ezin dela erakusten duen kaptura:



Hainbat saiakera egin ondoren, ez dugu lortu hau konpontzea, beraz .parquet fitxategia eskuz igotzea erabaki dugu impalara ‘upload columnar file to database’ aginduarekin eta hemendik grafikak sortzeko asmoarekin.

Uste genuen horrela lortuko genuela baina, ezin izan dugu probatu izan ere, datuak Impalara eskuz igotzerakoan ondorengo errorea ematen zigun terminalean.

superset_app | File "/app/superset/models/core.py", line 937, in _has_view
superset_app | view_names = dialect.get_view_names(connection=conn, schema=schema)
superset_app | File "/usr/local/lib/python3.8/site-packages/sqlalchemy/engine/interfaces.py", line 332, in get_view_names
superset_app | raise NotImplementedError()
superset_app | NotImplementedError
superset_app | 2023-05-01 21:13:28,358:WARNING:superset.models.core:Has view failed
superset_app | Traceback (most recent call last):

superset_app | File "/app/superset/models/core.py", line 937, in _has_view
superset_app | view_names = dialect.get_view_names(connection=conn, schema=schema)
superset_app | File "/usr/local/lib/python3.8/site-packages/sqlalchemy/engine/interfaces.py", line 332, in get_view_names
superset_app | raise NotImplementedError()
superset_app | NotImplementedError

Ez gara gai izan hau konpontzeko, hainbat aldiz probatu dugu datuak igotzea baina denbora askoz igotzen egon eta gero ,behin eta berriz errore berdinarekin topatu gara..

Dena dela, jadanik pentsaturik geneuzkan sortu beharreko grafikoak zeintzuk izango ziren eta nahiz eta ezin erakutsi, behintzat aipatu egingo ditugu.

- **Barra-diagrama:** barra-diagrama bat eraikitzea asteko egunaren edo eguneko orduaren araberako bidaia kopurua erakusteko.

- **Lerro-diagrama:** lerro-diagrama bidez denboran zehar egindako bidaia kopurua erakustea.

- **Bero-mapa:** bero-mapa bat sortzea hirian bidaia gehien egiten diren lekuak erakusteko.

- **Diagrama zirkularra:** diagrama zirkularra erabiltzea ibilgailu motaren edo ordainketa motaren araberako bidaien banaketa erakusteko.

- **Sakabanatze grafikoa:** sakabanaketa grafikoa erabilgarria izan daiteke bidaia-denboraren eta egindako distantziaren arteko erlazioa erakusteko.

- **Kutxa eta bibote grafikoa:** Kutxa eta bibote grafikoa erabilgarria izan daiteke bidaia-denboraren edo egindako distantziaren banaketa erakusteko.

- **Histograma:** Bidaien iraupenen banaketa adierazten duen histograma.

GEHIGARRIAK

Behin oinarrizko atazak burututa, gehigarri batzuk egiten ahalegindu gara:

Azureko hedapena

Azureko hedapena egiteko, honako pausoak eman ditugu: Lehenik eta behin, ‘grupo de recursos’ bat sortu dugu honako komandoarekin:

‘az group create --name lana2 --location northeurope ‘

Ondoren, kontainer biltegi bat sortu dugu, aurreko errekurtso taldean, komando honekin:

‘az acr create --resource-group lana2 --name dmpabigarrenlana --sku basic’

Hona igoko ditugu erabilitako irudi guztiak, lehenik eta behin lehenik eta behin izena aldatuz:

‘docker tag apache/kudu:latest dmpabigarrenlana.azurecr.io/kudu:latest’

eta ondoren push eginez:

‘docker push dmpabigarrenlana.azurecr.io/kudu‘

Hau, erabilitako irudi bakoitzarentzat egin dugu, eta hemen ez ezik, ‘.yml’ fitxategietan ere images atalean izenak aldatu dizkiegu, entregako azure karpetako ‘.yml’ fitxategietan ikus daitekeen bezela. Behin hau dena eginda, eta azuren login eginda, proiektua abiarazteko agindua eman dugu. Baina dirudienez, ez dugu baliabide nahikoa hau aurrera emateko, honako mezua lortu baitugu:

‘containerinstance.ContainerGroupsClient#CreateOrUpdate: Failure sending request: StatusCode=400 -- Original Error: Code="ContainerGroupQuotaReached" Message="Resource type 'Microsoft.ContainerInstance/containerGroups' container group quota 'StandardCores' exceeded in region 'northeurope'. Limit: '6', Usage: '0' Requested: '6.01'."’

Baliabide muga dela eta, ezin izan dugu gure proiektua azuren hedatu.

ONDORIOAK

Nahiz eta pauso guztiak klasean edota tutorialetan ikusi bezala eman ditugun, oso zaila egin zaigu aurreratzea, hainbat errore desberdinekin topatu gara proiektuan zehar, gehienak soluzionatu ditugu baina superseteko bi errore horiek dirudienez honen implementazioko akatsak dira eta ez dugu ia dokumentaziorik bilatu. Hortaz, ezin izan dugu proiektua osotasunean garatu, panelaren sorkuntza egin gabe utziz. Hala ere, datu basearen sorkuntza, eta kontainerren arteko konexio guztiak ondo doaztela frogatu dugu.

Kode guztia github errepositorio honetan dago, ezin bainuen igo egelara.
https://github.com/manexsora/dmpa