Assignment #5

გასაკეთებელია airflow pipeline, რომელმაც გარკვეული ინტერვალებით უნდა უზრუნველყოს pokemon.csv ფაილიდან რეპორტის გენერაცია.
უფრო დეტალურად, Pipeline შემდეგი ნაბიჯებისა და კომპონენტებისგან უნდა შედგებოდეს:

1) FakeStream_Dag

საწყის ეტაპზე უნდა შექმნათ ორი დაგი, რომელიც პარალელურად გაეშვება და pokemon1.csv და pokemon2.csv ფაილებს წაიკითხავს.
თითოეული დაგი ამ ფაილებიდან რენდომზე შეარჩევს ერთ row-ს და თითო ფაილად ჩაწერს DataLake ფოლდერში HDFS-ზე.
ამ ფაილების სახელის pattern და ფორმატი სურვილისამებრ შეარჩიეთ.

ეს დაგები რენდომ ფაილებს უნდა აგენერირებდნენ ხშირ-ხშირად, დაახლოებით 10-30 წმ-ის ინტერვალებით.
მონაცემებზე და ფაილებზე მანიპულაციის ტექნოლოგიაში არ ხართ შეზღუდული (spark, pandas, python ...).

2) DLToStaging_Dag

შემდეგ ეტაპზე უნდა შექმნათ მორიგი ორი დაგი. თითოეული დაგი hdfs სენსორის საშუალებით უნდა ამონიტორინგებდეს
იმ ფაილებს, რომელიც ჩაწერა თქვენმა FakeStream-ის პროცესებმა (dag-ებმა). ეს ფაილები სენსორმა უნდა აღმოაჩინოს regex-ით,
რომელიც შეესაბამება ამ ფაილების სახელის პატერნს.

სენსორის შესრულებისას უნდა დასტარტოთ პროცესი, რომელიც აღნიშნულ ფაილებს წაიღებს და ჩაწერს staging დირექტორიაში,
ისევ და ისევ HDFS-ზე. თქვენზეა დამოკიდებული, სთეიჯინგში ამ მონაცემებს როგორ შეინახავთ.

3) FiteredData_Dag

შემდეგი ეტაპზე უნდა შექმნათ კიდევ ერთი დაგი, რომელიც დაელოდება წინა dag_run-ების სიმრავლის დასრულებას (DLToStaging_Dag-ებს).
შემდეგ staging ფოლდერში არსებული ფაილების საფუძველზე შექმნის ცხრილს და ჩაწერს ფაილებიდან წაკითხულ ინფორმაციას.
გაითვალისწინეთ, რომ FakeStream მა შეიძლება დააგენერიროს დუბლიერბული მონაცემები და საბოლოო ცხრილში/ბაზაში უნიკალურები უნდა დატოვოთ.

ამოცანის ამ ეტაპზე უნდა გამოყენოთ ExternalTaskSensor, რომლის გადაკეთება, გადაწერა მოგიწევთ, რადგან არსებული იმპლემენტაცია მხოლოდ ერთ სხვა დაგში არსებული ტასქის დარულებას ელოდება, ამოცანაში კი გჭირდებათ, რომ სენსორი ელოდებოდეს ორი დაგის შესრულებას.

4) PokemonReport_Dag

ამ დაგში უნდა მოამზადოთ წინა სთეფებზე უკვე შექმნილი და ცხრილად ჩაწერილი ინფორმაციის მიხედვით რეპორტი.
ქვემო დეტალურადაა აღწერილი როგორ გააკეთოთ რეპორტი.

სხვა დეტალები დაგებთან დაგავშირებით:

FakeStream_pokemon1_dag-ს უნდა გაუწეროთ schedule, რომელიც 10-30 წამის სიხშირით გაეშვება.

FakeStream_pokemon2_dag-ს უნდა გაუწეროთ schedule, რომელიც 10-30 წამის სიხშირით გაეშვება.

DLToStaging1_dag ეს დაგი დაისტარტება თრიგერით FakeStream_pokemon1_dag დან. დამოუკიდებელი schedule არ აქვთ.

DLToStaging2_dag ეს დაგი დაისტარტება თრიგერით FakeStream_pokemon2_dag დან. დამოუკიდებელი schedule არ აქვთ.

FilteredData_dag დაგი უნდა ელოდებოდეს სენსორის საშუალებით DLToStaging1_dag, DLToStaging2_dag დაგებს, მაგრამ თავისი schedule-იც უნდა ჰქონდეს, რომელიც ყოველ 2 წუთში დასტარტავს დაგს.

PokemonReport_Dag დაგი უნდა ეშევბოდეს 15 წუთის სიხშირით. არ აქვს არც სენსორი და არც თრიგერი. მხოლოდ schedule.

დაგების მიმართება:

```
FakeStream_pokemon1_dag >> trigger >> DLToStaging1_dag (/2Min)
                                                          >> sensor >>  FilteredData_dag  >> /15Min >>  PokemonReport_Da
FakeStream_pokemon2_dag >> trigger >> DLToStaging1_dag (/2Min)
```

დავალების შესასრულებლად უნდა გამოიყენოთ დაგების დინამიურად გენერაციის ფუნქციონალი. ამისათვის უნდა შექმნათ რესურს ფაილი (კონფიგურაციის ფაილი, მაგ: yaml, json ...) დაგებისთვის, სადაც აღწერილი იქნება ის მეტა ინფორმაცია, რაც დაგებისა და თასქების არსებობისთვის არის აუცილებელი.

რესურს ფაილების შიდა სტრუქტურის მოწყობა თქვენ გადაწყვიტეთ, მთავარია ფაილების დაპარსვის შედეგად აღნიშნული დაგები და flow მივიღოთ.

ტრანსფორმაციის ლოგიკა

ველი	გამოყენება
classification **type2** **type1**	ამ ველებიდან ***ერთ-ერთის*** მიხედვით ვაჯგუფებთ, ორივე აუცილებელი არაა. Default-ად ერთ ერთი შეარჩიეთ. თუმცა ამ პარამეტრის გადაცემა DAG-ის და-trigger-ების დროსვე უნდა შემეძლოს (DAG trigger w/config)
	ზემოთ შერჩეული ერთი ველის მიხედვით უნდა გააკეთოთ შემდეგი ოპერაციები და აგრეგაციები:
japanese_name **experience_growth** **percentage_male** **pokedex_number** **generation**	ეს ველები არ გვჭირდება საერთოდ
name	ცალკე ველად არ გვჭირდება, მაგრამ საჭირო ველებისთვის არის აღწერილი როგორ უნდა გამოიყენოთ
is_legendary	დავთვალოთ რამდენი legendary pokemon გვხვდება თითოეულ type-ში
abilities	დავთვალოთ ჯამში რამდენი განსხვავებული ability გვხვდება თითოეულ type-ში
against_bug **against_dark** **against_dragon** **against_electric** **against_fairy** **against_fight** **against_fire** **against_flying** **against_ghost** **against_grass** **against_ground** **against_ice** **against_normal** **against_poison** **against_psychic** **against_rock** **against_steel** **against_water**	თითოეულისთვის ვითვლით ამ ველის ჯამს

ველი	გამოყენება
attack	თითოეული ამ ველის საშუალო, მინიმალური და მაქსიმალური. Min-სა და max-ს იგივე ველში მივუწეროთ რომელ პოკემონს აქვს ეს სტატისტიკა (Name ველი). თუ რამდენიმე პოკემონს აქვს, რომელიმე ერთს ვირჩევთ
base_egg_steps	
base_happiness	
base_total	
capture_rate	
defense	
height_m	
hp	
sp_attack	
sp_defense	
speed	
weight_kg	
total	ამ ველს თქვენ დააგენერირებთ და ბოლოში მიადგამთ თქვენს რეპორტს. ლოგიკა შემდეგია: თითოეული სვეტისთვის დაითვლით გლობალურ ჯამს (სადაც ჯამია. საშუალოებზე - საშუალოს, min-ზე - min-ს და max-ზეც შესაბამისად)

აუცილებელი პირობა: შეგიძლიათ გამოიყენოთ სპარკი ან პანდასი, რომელიც გაგიხარდებათ. ან პირდაპირ Python-ზე გააკეთოთ. როგორც ნახეთ/ნახავთ, რეპორტი უამრავი ველისგან შედგება და მათზე აგრეგაციებს ვითხოვთ. თუმცა ეს აგრეგაციები მრავალფეროვანი არაა და ბევრი ველისთვის ერთი და იგივეა. ამიტომ, აუცილებელი მოთხოვნაა, რომ დავალების ეს ნაწილიც მაქსიმალურად დინამიურად და შედარებით განზოგადებულად გააკეთოთ. ვგულისხმობთ, რომ, მაგალითად, ჯამის დათვლა თითოეული ველისთვის ცალ-ცალკე ხელით და copy-paste-ით არ უნდა წეროთ. აგრეგაციების ლოგიკა განაზოგადეთ. შეგიძლიათ გქონდეთ რაიმე კონფიგი, სადაც და-map-ავთ, თითოეულ ველს რომელი ტრანსფორმაცია შეესაბამება. შეგიძლიათ შემოგვთავაზოთ ალტერნატიული იმპლემენტაციაც. მთავარია, შეძლებისდაგვარად განზოგადებული იყოს.

ბონუსი

1) ბონუს დავალების შესასრულებლად უნდა გამოიყენოთ დაგების დინამიურად გენერაციის ფუნქციონალი. ამისათვის უნდა შექმნათ რესურს ფაილი,(კონფიგურაციის ფაილი, მაგ: yaml, json ...) დაგებისთვის, სადაც აღწერილი იქნება ის მეტა ინფორმაცია, რაც დაგებისა და თასქების არსებობისთვის არის აუცილებელი.რესურს ფაილების შიდა სტრუქტურის მოწყობა თქვენ გადაწყვიტეთ, მთავარია ფაილების დაპარსვის შედეგად დავალების ძირითადი ნაწილის შესაბამისი დაგები და flow მივიღოთ.

2) დაწერეთ ოპერატორი, რომელიც მოაგროვებს მეტა ინფორმაციას Airflow ს ბაზაში, dag_run ცხრილიდან, მიმდინარე dag_run ის ჭრილში. ოპერატორს უნდა გადაეცემოდეს იმ ოპერატორის ტიპი, რომლის მეტა ინფორმაციაც უნდა წაიკითხოს და დაბეჭდოს (მაგ: PythonOperator). ოპერატორი უნდა აბრუნებდეს dag_run-ის ჭრილში, გადაცემული ოპერატორის ტიპის მიხედვით, სრულ მეტა ინფორმაციას, რაც აღნიშნულ ცხრილში აქვს, რომელიც გაფილტრული უნდა იყოს სტატუსით (Success ან Fail). ანუ საჭიროა დაბეჭდოთ ორი სიმრავლე მეტა ინფორმაციის ცალკე Fail თასქების სიმრავლე და ცალკე Success თასქების სიმრავლე.