

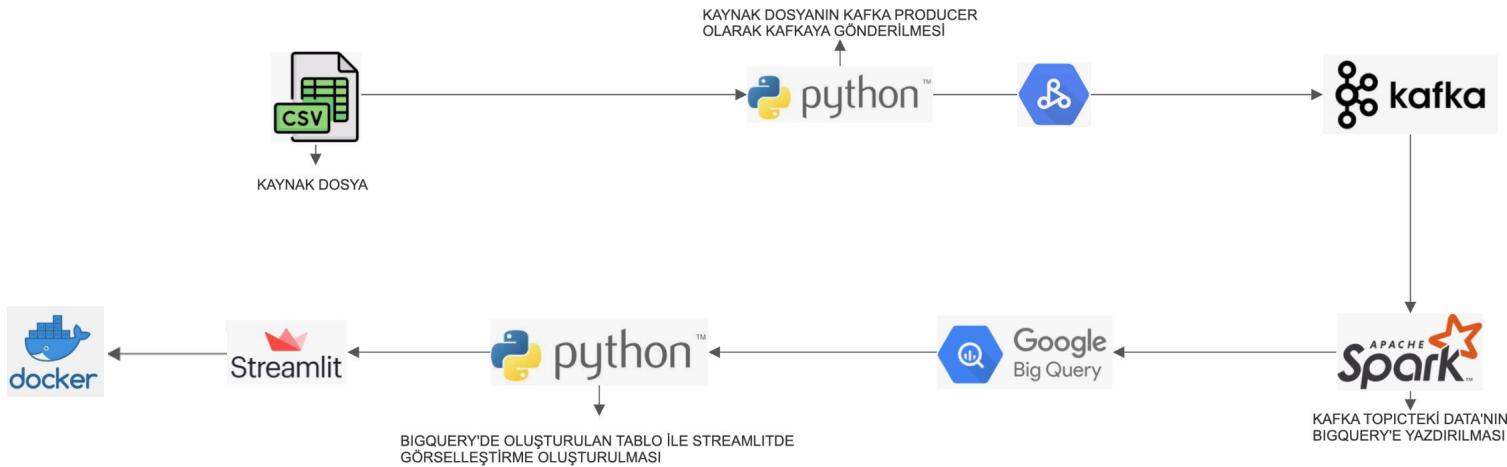
Kerem NOGAY

Kafka-SPARK Proje Sunum



Proje Özeti

New York şehrindeki köpek saldırılarına ait **DATASET**'i kullanılarak **DATAPROC** üzerinde **KAFKA**'ya mesaj gönderilmesi ve **SPARK** aracılığı ile **BIGQUERY**'de oluşturulan tablonun **STREAMLIT** ile görselleştirilerek **DOCKER** ile **DEPLOY** edilmesi.



Ortam Kurulumu



DATAPROC

Öncelikle projeyi gerçekleştireceğimiz ortamı hazırlayarak başlayacağız. **GCP** üzerinde **DATAPROC**'a gelerek **COMPUTE ENGINE** ile yeni bir **CLUSTER** oluşturuyoruz. **CLUSTER**'ı default ayarları kullanarak oluşturuyoruz fakat burada oluşturduğum **CLUSTER**'ı bir web arayüzden monitör edebilmek için **Enable component gateway** seçeneğini seçiyorum. Gerekli olması halinde kullanmak üzere **Jupyter Notebook**'u da kurulumu ekledim.

Create Dataproc cluster
Select the infrastructure service that you want to use.

Cluster on Compute Engine
Create the cluster on Compute Engine. **CREATE**

Cluster on GKE
Create the cluster on Google Kubernetes Engine (GKE). **CREATE**

CREATE CLUSTER

Create a Dataproc cluster on Compute Engine

- Set up cluster
Begin by providing basic information.
- Configure nodes (optional)
Change node compute and storage capabilities.
- Customize cluster (optional)
Add cluster properties, features, and actions.
- Manage security (optional)
Change access, encryption, and security settings.

Autoscaling
Automates cluster resource management based on an autoscaling policy.
Policy: None

Enhanced Flexibility Mode
Dataproc Enhanced Flexibility Mode (EFM) manages shuffle data to minimize job progress delays caused by the removal of nodes from a running cluster. EFM offloads shuffle data in one of two user-selectable modes, primary worker shuffle and Hadoop Compatible File System (HCFS) shuffle. [Learn more](#)

An autoscaling policy must be selected to configure EFM.

Versioning
Use a custom image to load pre-installed packages. [Learn more](#)

Image Type and Version
2.0-debian10

Release Date
First released on 1/22/2021.

Components
Component Gateway
 Enable component gateway
Provide access to the web interfaces of default and selected optional components on the cluster. [Learn more](#)

Optional components
Select one or multiple components. [Learn more](#)

- Anaconda [?](#)
- Hive WebHCat [?](#)
- Jupyter Notebook [?](#)
- Zeppelin Notebook [?](#)
- Druid [?](#)
- Presto [?](#)
- ZooKeeper [?](#)
- Ranger [?](#)
- HBase [?](#)
- Flink [?](#)
- Docker [?](#)
- Solr [?](#)

Ortam Kurulumu



DATAPROC

CLUSTER aktif hale geldikten sonra cluster içerisinde **CLOUD SHELL** kullanarak **SSH** bağlantısı yapacağım.

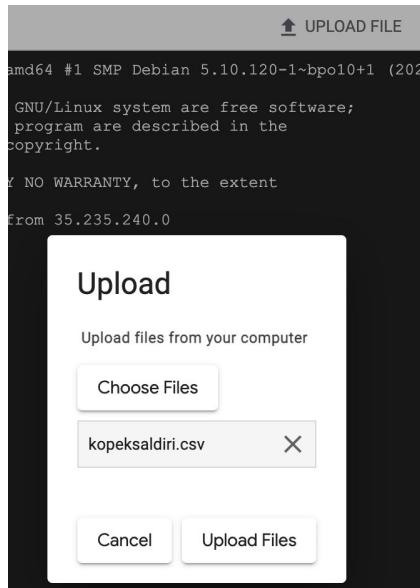
Jobs on Clusters									
		Filter Search clusters, press Enter							
Clusters		Name	Status	Region	Zone	Total worker nodes	Scheduled deletion	Cloud Storage staging bucket	Created
<input type="checkbox"/>	keremkafka	Running	us-central1	us-central1-b	2	Off	dataproc-staging-us-central1-728117702780-ju88stqn	Oct 17, 2022, 7:09:48 PM	

MONITORING	JOB	VM INSTANCES	CONFIGURATION	WEB INTERFACES	
<input type="checkbox"/>	Name	katkaprojesim			SSH

Kaynak Dosya



Kurulumlara başlamadan önce üzerinde çalışacağımız **CSV** dosyasını sunucuya yükledim. Projeye kaynak olarak oluşturduğum **kopeksaldiri.csv** dosyası ile devam edeceğim.



1	01.01.2018	DOG	UNKNOWN	UNKNOWN	U	false	Brooklyn	11220
2	01.04.2018	DOG	UNKNOWN	UNKNOWN	U	false	Brooklyn	UNKNOWN
3	01.06.2018	DOG	Pit Bull	UNKNOWN	U	false	Brooklyn	11224
4	01.08.2018	DOG	UNKNOWN	4	M	false	Brooklyn	11231
5	01.09.2018	DOG	Pit Bull	UNKNOWN	U	false	Brooklyn	11224
6	01.03.2018	DOG	BASENJI	4Y	M	false	Brooklyn	11231
7	01.01.2018	DOG	UNKNOWN	UNKNOWN	U	false	Brooklyn	UNKNOWN
8	01.03.2018	DOG	Pit Bull	UNKNOWN	U	false	Brooklyn	11233
9	01.04.2018	DOG	Pit Bull	5Y	M	false	Brooklyn	11235

Gerekli Kurulumlar



DATAPROCTA cluster içerisinde **SSH** bağlantısı yaptıktan sonra **KAFKA**'yı sitesinden aldığım bağlantı ile indirerek arşivden çıkarttım. Ardından **KAFKA PYTHON** paketini kuracağım.

WGET

```
wget https://downloads.apache.org/kafka/3.3.1/kafka\_2.12-3.3.1.tgz
```

EXTRACT

```
tar -zvxf kafka_2.12-3.3.1.tgz
```

KAFKA-PYTHON

```
pip install kafka-python
```

3.3.1

- Released October 3, 2022
- [3.3.1](#) and [3.3.0](#) Release Notes
- Source download: [kafka-3.3.1-src.tgz \(asc, sha512\)](#)
- Binary downloads:
 - Scala 2.12 - [kafka_2.12-3.3.1.tgz \(asc, sha512\)](#)
 - Scala 2.13 - [kafka_2.13-3.3.1.tgz \(asc, sha512\)](#)

Kafkada İlk Adımlar



KAFKA klasörü içerisinde **ZOOKEEPER** ve **KAFKA**'yı başlatıyoruz. Bunu sürekli arkada çalışan bir servis olarak çalışması için **NOHUP** ile yapacağız. Ardından proje için kullanacağımız **kafkaprojesi** isimli **KAFKA TOPIC**'ı oluşturuyoruz.

ZOOKEEPER

```
nohup bin/zookeeper-server-start.sh config/zookeeper.properties
```

KAFKA

```
nohup bin/kafka-server-start.sh config/server.properties
```

TOPIC

```
bin/kafka-topics.sh --create --topic kafkaprojesi --bootstrap-server localhost:9092
```

Producer Hazırlanması



Burada daha önce **CSV** olarak yüklediğim **DATASET** ile **KAFKA**'ya mesaj gönderiyormuş gibi bir **SENARYO** oluşturuyorum. **PY** dosyasında **KAFKA TOPIC**'ı ve ilgili **CSV** dosyasını tanımladım. Oluşturduğum **PY** dosyasını sunucuya **UPLOAD** ederek çalıştırıldığında her 5 saniyede bir **CSV** dosyasından 10 satır olarak **KAFKA** da tanımladığım **TOPIC**'e mesaj gönderiyor.

```
from kafka import KafkaProducer      #kafka paketini import ediyor
from json import dumps
import json                         #kafkaya verileri json olarak göndereceğiz
import csv                          #dosyayı okumak için csv modülünü kullandım
import time                         #sleep kullanmak için time modülünü import ettim
from itertools import islice        #dosya içerisindeki satırları belli bir sayıda okumak için itertools islice kullandım

if __name__ == "__main__":
    # execute only if run as a script

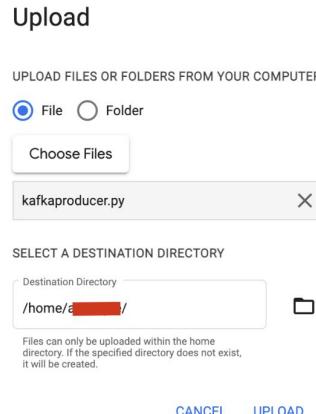
    #bir while döngüsü içerisinde csv dosyasını açıyorum
    #producer ile kafka producer oluşturuyorum utf-8 ve json formatında datayı oluşturuyorum

    with open('kopeksaldırı.csv') as file:
        producer = KafkaProducer(value_serializer=lambda m: json.dumps(m).encode('utf-8'))
        reader = csv.reader(file, delimiter=",")

    #reader ile okuduğum dosyayı islice ile 10 satır alacak şekilde next_n_lines'a atıyorum
    #if not ile bunun boş olup olmadığına bakıyorum boş değilse her line için producer.send ile
    #verileri sözlüğe çevirerek ilgili topic'e gönderiyorum

    while True:
        next_n_lines = islice(reader, 10)
        if not next_n_lines:
            break
        else:
            for line in next_n_lines:
                producer.send('kafkaprojesi', {'id': line[0], 'tarih': line[1], 'cins': line[3], 'bolge': line[7]})

            producer.flush() #commit ediyor
            time.sleep(5)
```



Producer Hazırlanması

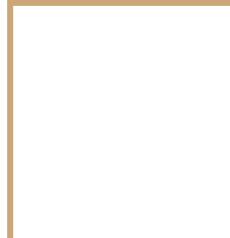


KAFKA üzerinde oluşturduğumuz **TOPIC**'e **PRODUCER**'dan veri gelip gelmediğini kontrol ediyoruz.
Aşağıdaki gibi **TOPIC**'e mesajlar geldiğini görebiliriz.

READ TOPIC

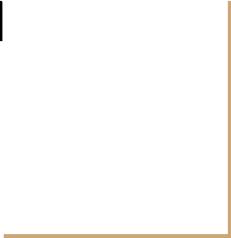
```
bin/kafka-console-consumer.sh --topic kafkaprojesi --from-beginning --bootstrap-server localhost:9092
```

```
:~/kafka_2.13-3.3.1$ bin/kafka-console-consumer.sh --topic kafkatopic2 --from-beginning  
--bootstrap-server localhost:9092  
{"id": "1", "tarih": "01.01.2018", "cins": "UNKNOWN", "bolge": "Brooklyn"}  
{"id": "2", "tarih": "01.04.2018", "cins": "UNKNOWN", "bolge": "Brooklyn"}  
{"id": "3", "tarih": "01.06.2018", "cins": "Pit Bull", "bolge": "Brooklyn"}  
{"id": "4", "tarih": "01.08.2018", "cins": "UNKNOWN", "bolge": "Brooklyn"}  
{"id": "5", "tarih": "01.09.2018", "cins": "Pit Bull", "bolge": "Brooklyn"}  
{"id": "6", "tarih": "01.03.2018", "cins": "BASENJI", "bolge": "Brooklyn"}  
{"id": "7", "tarih": "01.01.2018", "cins": "UNKNOWN", "bolge": "Brooklyn"}  
{"id": "8", "tarih": "01.03.2018", "cins": "Pit Bull", "bolge": "Brooklyn"}  
{"id": "9", "tarih": "01.04.2018", "cins": "Pit Bull", "bolge": "Brooklyn"}  
{"id": "10", "tarih": "01.10.2018", "cins": "UNKNOWN", "bolge": "Brooklyn"}  
{"id": "11", "tarih": "01.06.2018", "cins": "UNKNOWN", "bolge": "Brooklyn"}  
{"id": "12", "tarih": "01.07.2018", "cins": "Yorkshire Terrier Crossbreed", "bolge": "Brooklyn"}
```



SPARK

CONSOLE STREAM



SPARK SÜREÇLERİ



CONSOLE STREAM-1

KAFKA 'ya mesaj olarak gönderdiğimiz **DATA**'yı ilgili **TOPIC**'ten **SPARK** ile okumak için önce **SPARK SHELL**'i kafkaya uyumlu olacak şekilde çalıştırıyoruz. Ardından bağlantı bilgilerini set ederek **SCALA** ile **KAFKA** bağlantısı yapıyoruz. Bağlantı yaptıktan sonra **SPARK** üzerinde kullanacağımız paketleri **IMPORT** ediyoruz.

SPARK SHELL BAŞLATMA

```
spark-shell --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2
```

KAFKA BAĞLANTISI (Shuffle datayı executorlar veya worker nodelar arasında taşıyor. Stream datayı okumak için spark.readStream metodunu kullandık)

```
spark.conf.set("spark.sql.shuffle.partitions",4)
```

```
val kafkadf = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("subscribe", "kafkaprojesi").load
```

IMPORTS

```
import org.apache.spark.sql.types._ (SCHEMA oluşturmadan önce import ediyorum)
```

```
import org.apache.spark.sql.streaming.Trigger.ProcessingTime (Bunu stream işlemini başlatmak için kullanıyoruz)
```

SPARK SÜREÇLERİ



CONSOLE STREAM-2

Öncelikle bir **SCHEMA** oluşturuyoruz. **BASEDF** ile veriyi okuyarak **TOPIC**'e gelen mesajları oluşturduğum **SCHEMA** ile birleştirerek hesaplamalar yapacağım. **SCHEMA** da bulunan tarih, bölge ve cins alanlarını groplayarak **COUNT** alıyorum.

SCHEMA

```
val dfschema = StructType( List( StructField("id", IntegerType),StructField("tarih",StringType), StructField("cins",StringType),StructField("bolge",StringType)))
```

BASEDF (**BASEDF** ile verileri string olarak aldım tarih alanını raporlama sırasında Date olarak kullanacağım)

```
val basedf = kafkadf.select(from_json($"value".cast("string")), dfschema).alias("dfalias"))
```

GROUP BY

```
val countdf = basedf. groupBy($"dfalias"("tarih"),$"dfalias"("bolge"),$"dfalias"("cins")).count. sort($"count".desc)
```

SPARK SÜREÇLERİ



CONSOLE STREAM-3

SPARK üzerinde hazırladığım **SCHEMA** ve **DATAFRAME'i** **CONSOLE** üzerinde **STREAM** olarak test etmek için daha önce **IMPORT** ettiğim kütüphaneyi kullanacağım. **STREAMCOUNTDF** çalıştırıktan sonra stream veriyi hesaplamaya başlıyor.

STREAM (Console'a yazmak istediğimiz için **console** olarak belirttik. Mesajları silmesini istemediğimiz için **Truncate, false** kullandık. 5 saniyede bir güncelleyecek)

```
val streamcountdf = countdf.writeStream.outputMode("complete").format("console").option("truncate","false").trigger(ProcessingTime("5 seconds")).start
```

```
-----  
Batch: 3  
-----  
+---+---+---+---+  
|dfalias[tarih] |dfalias[bolge] |dfalias[cins] |count|  
+---+---+---+---+  
|01.01.2018    |Brooklyn      |UNKNOWN      | 3   |  
|03.17.2018    |Brooklyn      |UNKNOWN      | 2   |  
|01.29.2018    |Brooklyn      |UNKNOWN      | 2   |
```

BİLGİ: streamcountdf.stop dediğimizde durdurabiliriz.

SPARK SÜREÇLERİ

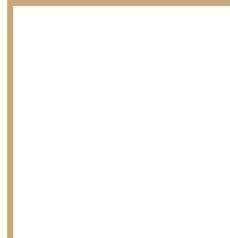


CONSOLE STREAM FINAL

Projeyi **CONSOLE** 'da **STREAM** ettik. Sonuç olarak oluşan kod şu şekilde olacaktır. Bunu scala dosyası olarak kaydedip şu şekilde terminalde başlatabiliriz.

```
spark-shell --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 -i consolestream.scala
```

```
import org.apache.spark.sql.types._  
import org.apache.spark.sql.streaming.Trigger.ProcessingTime  
  
val kafkadf = spark.readStream.format("kafka"). option("kafka.bootstrap.servers", "localhost:9092"). option("subscribe", "kafkaprojesi").load  
  
val dfschema = StructType( List( StructField("id", IntegerType),StructField("tarih",StringType), StructField("cins",StringType),StructField("bolge",StringType)))  
  
val basedf = kafkadf.select(from_json($"value".cast("string"), dfschema).alias("dfalias"))  
  
val countdf = basedf. groupBy($"dfalias"("tarih"),$"dfalias"("bolge"),$"dfalias"("cins")).count. sort($"count".desc)  
  
val streamcountdf = countdf. writeStream.outputMode("complete").format("console"). option("truncate","false"). trigger(ProcessingTime("5 seconds")).start
```



SPARK

BIGQUERY WRITE DATA



SPARK SÜREÇLERİ



BIGQUERY-1

Şimdi **SPARK** ile **TOPIC**'ten gelen veriyi **BIGQUERY**'e yazdıracağız. Daha önceki adımlarda **KAFKA** bağlantısını nasıl yaptığımızı belirtmiştim. Gene aynı **DATAFRAME** ve **SCHEMA**'lar ile devam edeceğim. **SPARK** ile **TOPIC**'ten gelen veriyi **BIGQUERY**'e yazdıracağız. Öncelikle **BIGQUERY** değişkenlerini tanımladım. **BUCKET** ile **DATAPROC** aynı **REGION** da olmalı.

SPARK (Sparkı Bigquery ile başlattım)

```
spark-shell --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 --jars=gs://spark-lib/bigquery/spark-bigquery-latest_2.12.jar
```

KAFKA BAĞLANTISI

```
val kafkadf = spark.readStream.format("kafka").option("kafka.bootstrap.servers", "localhost:9092").option("subscribe", "kafkaprojesi").load
```

IMPORT

```
import org.apache.spark.sql.types._ (SCHEMA oluşturmadan önce import ediyorum)
```

```
import org.apache.spark.sql.streaming.Trigger.ProcessingTime (Bunu stream işlemini başlatmak için kullanıyoruz)
```

BIGQUERY TANIMLARI

```
val bucket = "keremkafkabucket"
```

```
spark.conf.set("temporaryGcsBucket", bucket)
```

```
spark.conf.set("parentProject", "keremkafka")
```

SPARK SÜREÇLERİ



BIGQUERY-2

Daha önce oluşturduğum **SCHEMA**'yı tekrar kullanıyorum. **BASEDF** ile veriyi okuyarak **COUNTDF** ile gruplama yaparak daha sonra veriyi **TABLO**'ya yazdıracağız.

SCHEMA TANIMLAMASI

```
val dfschema = StructType( List( StructField("id", IntegerType),StructField("tarih",StringType), StructField("cins",StringType),StructField("bolge",StringType)))
```

BASEDF

```
val basedf = kafkadf.select(from_json($"value".cast("string")), dfschema).alias("dfalias"))
```

GROUP BY

```
val countdf = basedf. groupBy($"dfalias"("tarih"),$"dfalias"("bolge"),$"dfalias"("cins")).count. sort($"count".desc)
```

SPARK SÜREÇLERİ



BIGQUERY-3

Veriyi yazdırınmak için **BIGQUERY**'de tablo oluşturacağım. Simdilik tarih alanını da **STRING** olarak oluşturuyorum raporlama sürecinde **DATE** olarak kullanacağım. Ardından **IAM** de **BIGQUERY**'e veri yazabilmek için servis hesabı oluşturup **KEY** dosyasını terminalden yükledim.

keremkafka

External connections

keremkafka

kopeksaldırı

Filter Enter property name or value

	Field name	Type	Mode
<input type="checkbox"/>	TARIH	STRING	NULLABLE
<input type="checkbox"/>	CINS	STRING	NULLABLE
<input type="checkbox"/>	BOLGE	STRING	NULLABLE
<input type="checkbox"/>	VAKA	INTEGER	NULLABLE

EDIT SCHEMA **VIEW ROW ACCESS POLICIES**

Assign roles

Roles are composed of sets of permissions and determine what the principal can do with this resource. [Learn more](#)

Role	IAM condition (optional)	Actions
BigQuery Admin	+ ADD IAM CONDITION	
Cloud SQL Admin	+ ADD IAM CONDITION	
Editor	+ ADD IAM CONDITION	
Cloud Run Admin	+ ADD IAM CONDITION	

+ ADD ANOTHER ROLE

SPARK SÜREÇLERİ



BIGQUERY-4

KEY dosyasını **CREDENTIALS** olarak tanımladım ve **writeStream** ile **BIGQUERY**'de oluşturduğum **DATASET** altındaki **TABLO**'ya yazdırıldım.

TABLOYA YAZMA

```
val dfcountquery = countdf.writeStream.outputMode("complete").format("bigquery").option("credentialsFile", "/home/kerem/keremkafka.json").option("table","keremkafka.kopeksaldiri").option("checkpointLocation", "path/to/checkpoint/dir/in/hdfs").option("failOnDataLoss",false).option("truncate",false).start().awaitTermination()
```

```
1 select *
2 from `keremkafka.keremkafka.kopeksaldiri` a
3 Order by a.dfalias_tarih_ asc
```

Query results

Row	dfalias_tarih_	dfalias_bolge_	dfalias_cins_	count
1	01.01.2018	Brooklyn	Pit Bull	9
2	01.01.2018	Brooklyn	West High White Terrier	9
3	01.01.2018	Brooklyn	UNKNOWN	25
4	01.03.2018	Brooklyn	BASENJI	9
5	01.03.2018	Brooklyn	Pit Bull	9
6	01.04.2018	Brooklyn	UNKNOWN	9
7	01.04.2018	Brooklyn	Pit Bull	9
8	01.06.2018	Brooklyn	UNKNOWN	9

SPARK SÜREÇLERİ

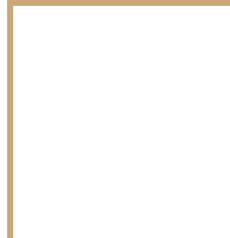


BIGQUERY FINAL

Projeyi **BIGQUERY**'de ilgili **TABLO**'ya yazdirdık. Sonuç olarak oluşan kod şu şekilde olacaktır. **SPARK SHELL** ile direkt dosyayı çalıştırabiliriz. Burada **ReadStreamde - failOnDataLoss, false** yapmamın nedeni **PRODUCER**'i yeniden başlattığım zaman **OFFSET**'lerde uyusuzluk nedeniyle durması. Bunu engellemek için bu ayarı ekledim.

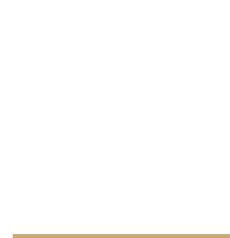
```
spark-shell --packages org.apache.spark:spark-sql-kafka-0-10_2.12:3.1.2 --jars=gs://spark-lib/bigquery/spark-bigquery-latest_2.12.jar -i bigquerywrite.scala
```

```
import org.apache.spark.sql.types._  
import org.apache.spark.sql.streaming.Trigger.ProcessingTime  
  
val bucket = "bucketadi"  
spark.conf.set("temporaryGcsBucket", bucket)  
spark.conf.set("parentProject", "projeadi")  
  
val kafkadf = spark.readStream.format("kafka").option("failOnDataLoss",false). option("kafka.bootstrap.servers", "localhost:9092"). option("subscribe", "topicadi").load  
  
val dfschema = StructType( List( StructField("id", IntegerType),StructField("tarih",StringType), StructField("cins",StringType),StructField("bolge",StringType)))  
  
val basedf = kafkadf.select(from_json($"value".cast("string"), dfschema).alias("dfalias"))  
  
val countdf = basedf. groupBy($"dfalias"("tarih"),$"dfalias"("bolge"),$"dfalias"("cins")).count. sort($"count".desc)  
  
val dfcountquery = countdf.writeStream.outputMode("complete"). format("bigquery")\  
.option("credentialsFile", "/home/kerem/dosyaadi.json")\  
.option("table","keremkafka.kopeksaldiri")\  
.option("checkpointLocation", "path/to/checkpoint/dir/in/hdfs")\  
.option("failOnDataLoss",false)\  
.option("truncate",false).start().awaitTermination()
```



STREAMLIT

VISUALIZATION



STREAMLIT



VISUALIZATION

STREAMLIT ile kullanmak üzere yeni bir servis hesabı açtım ve **BIG QUERY** data viewer yetkisi verdim. **STREAMLIT** ile kullanmak için **CREDENTIALS** dosyasını indirip format değişikliği yaptım. Ayarları **STREAMLIT** 'in kendi dökümantasyonundan devam ettirdim. sonraki slaytlarda **PYTHON** ile oluşturduğum görselleştirmenin kodlarını inceleyeceğiz.

The screenshot shows the Streamlit configuration interface. It includes fields for 'streamlit@keremkafka.iam.gserviceaccount.com' (Email), 'streamlit' (Service account name), 'No keys' (Key status), and '110052668902810673951' (Cloud ID). Below this, there are two sections: 'Add the key file to your local app secrets' and 'Add google-cloud-bigquery to your requirements file'.

Add the key file to your local app secrets

Your local Streamlit app will read secrets from a file `.streamlit/secrets.toml` in your app's root directory. Create this file if it doesn't exist yet and add the content of the key file you just downloaded to it as shown below:

```
# .streamlit/secrets.toml
[gcp_service_account]
type = "service_account"
project_id = "xxx"
private_key_id = "xxx"
private_key = "xxx"
client_email = "xxx"
client_id = "xxx"
auth_uri = "https://accounts.google.com/o/oauth2/auth"
token_uri = "https://oauth2.googleapis.com/token"
auth_provider_x509_cert_url = "https://www.googleapis.com/oauth2/v1/certs"
client_x509_cert_url = "xxx"
```

Add google-cloud-bigquery to your requirements file

Add the `google-cloud-bigquery` package to your `requirements.txt` file, preferably pinning its version (replace `x.x.x` with the version want installed):

```
# requirements.txt
google-cloud-bigquery==x.x.x
```

Write your Streamlit app

Copy the code below to your Streamlit app and run it. Make sure to adapt the query if you don't use the sample table.

```
# streamlit_app.py
import streamlit as st
from google.oauth2 import service_account
from google.cloud import bigquery

# Create API client.
credentials = service_account.Credentials.from_service_account_info(
    st.secrets["gcp_service_account"])
client = bigquery.Client(credentials=credentials)

# Perform query.
# Uses st.experimental_memo to only rerun when the query changes or after 10 min.
#st.experimental_memo(ttl=600)
def run_query(query):
    query_job = client.query(query)
    rows_raw = query_job.result()
    # Convert to list of dicts. Required for st.experimental_memo to hash the results.
    rows = [dict(row) for row in rows_raw]
    return rows

rows = run_query("SELECT word FROM `bigquery-public-data.samples.shakespeare` LIMIT 1000")

# Print results.
st.write("Some wise words from Shakespeare:")
for row in rows:
    st.write("• " + row['word'])
```

```
import streamlit as st
#google connectorlerini import ettim
from google.oauth2 import service_account
from google.cloud import bigquery
import pandas as pd
#grafikleri plotly ile oluşturdum
import plotly.express as px
#piechart için graph objects import ettim
import plotly.graph_objects as go
#sleep kullanmak için import ettim
import time
#numpy select kullandım
import numpy as np

# streamlit layout ayarları
st.set_page_config(
    page_title="NYC Kopek Saldırıları",
    layout="wide",
    initial_sidebar_state="auto",
)

# api clientin oluşturulması
credentials = service_account.Credentials.from_service_account_info(
    st.secrets["gcp_service_account"]
)
client = bigquery.Client(credentials=credentials)

#bu kısmı dökümantasyondan alarak projeme uyguladım
# clearcache ve rerun içinde bir bekleme süresi oluşturdu

@st.experimental_memo(ttl=10)
def run_query(sql):
    job_config = bigquery.QueryJobConfig(use_query_cache=False)
    query_job = client.query(sql, job_config=job_config)
    rows= query_job.to_dataframe()
    return rows

#sorgudan bir df oluşturuyorum
df=run_query("SELECT * FROM `keremkafka.kopeksaldiri`")

#rerun için manual bir buton ekledim
if st.button("Reload"):
    st.experimental_rerun()

#sütun adlarını yeniden isimlendirdim
df = df.rename(columns={"dfalias_tarih_":"Tarih",
                        "dfalias_bolge_":"Bolge",
                        "dfalias_cins_":"Cins",
                        "count" : "Vaka Sayisi"
                       })

#tarihi string geliyordu date'e çevirdim ve diğer alanlar içinde data typelerini tanımladım
df['Tarih'] = pd.to_datetime(df['Tarih']).dt.date

df= df.astype({
    "Bolge":str,
    "Cins":str,
    "Vaka Sayisi":int
})

#tarihten ay bilgisini alıp ayları yazdıracağım
aynumarasi = df['Ayno'] = pd.DatetimeIndex(df['Tarih']).month

#ayları isimlendiriyorum
durumlar = [
    (aynumarasi == 1),
    (aynumarasi == 2),
    (aynumarasi == 3),
    (aynumarasi == 4),
    (aynumarasi == 5),
    (aynumarasi == 6),
    (aynumarasi == 7),
    (aynumarasi == 8),
    (aynumarasi == 9),
    (aynumarasi == 10),
    (aynumarasi == 11),
    (aynumarasi == 12),
]

ayadları = ['Ocak','Subat','Mart','Nisan','Mayis','Haziran','Temmuz','Agustos','Eyluk','Ekim','Kasim','Aralik']

#numpy select ile durumlar ve ay adlarını tanımladım
df['Aylar'] = np.select(durumlar,ayadları)
```

```
##### Analizler buradan başlıyor #####
#sayfa başlığı ve açıklaması yazdım
st.title('NYC Köpek Saldırıları')
st.markdown("""Bolgelere ve cinslerine göre köpek saldırılı durumu""")
st.text("")

placeholder = st.empty()

#dökümantasyona göre grafiklerin interkatif olması için for ve while içerisinde
# yazılması önerilmiş bu şekilde 30snde bir yenileyecek

for seconds in range(30):

### Toplam Analizleri
#df den bölge ve cins alanlarının sayılarını alıyorum

    bolgesayisi= df["Bolge"].nunique()
    cinssayisi= df["Cins"].nunique()

#tekilsayı olarak yazdırıyorum
    with placeholder.container():
        bolge, cins, toplamv = st.columns(3)

        bolge.metric(
            label="Bölge Sayısı",
            value= bolgesayisi
        )
        toplamv.metric(
            label="Toplam Vaka",
            value= df['Vaka Sayısı'].sum()
        )
        cins.metric(
            label="Cins Sayısı",
            value= cinssayisi
        )

    #toplamlar ile grafikler arasında boşluk bırakmak için kullandım
    st.markdown("")
```

```
### Grafikler ###

# Bolgelere göre toplam saldırular. Bolgeye göre gruplandı ve azalan şekilde sıralandı ardından ilk 10 değer alındı.
#bolgedf
    toplambolge=df.groupby(['Bolge'])[['Vaka Sayısı']].sum().reset_index()
    toplambolgesort = toplambolge=toplambolge.sort_values(by=['Vaka Sayısı'],ascending=False)
    toplambolgedf = toplambolgesort.head(10)

# Bu degiskenleri ilk grafikte kullanacağım
    bolgelabel = toplambolgedf['Bolge']
    bolgevalues = toplambolgedf['Vaka Sayısı']

# Cinslere göre toplam saldırular(Cinslere göre gruplandı ve azalan şekilde sıralandı ardından ilk 10 değer alındı)
    toplamcins=df.groupby(['Cins'])[['Vaka Sayısı']].sum().reset_index()
    toplamcinssort=toplamcins.sort_values(by=['Vaka Sayısı'],ascending=False)
    toplamcinsdf = toplamcinssort.head(10)

#Bölge ve Cinsler beraber gruplandı ilk grafikte kullanacağım
    toplambolcins=df.groupby(['Bolge','Cins'])[['Vaka Sayısı']].sum().reset_index()
    toplambolcinssort=toplambolcins.sort_values(by=['Vaka Sayısı'],ascending=False)
    toplambolcinsdf = toplambolcinssort.head(10)
    toplambolcinsf = toplambolcinssort[toplambolcinssort['Vaka Sayısı'] > toplambolcinssort['Vaka Sayısı'].mean()]

#Ay adlarına göre grüplayarak toplam alırdıracığım bunu en alttaki detay tablodardan birinde göstereceğim
    toplamaylar=df.groupby(['Aylar'])[['Vaka Sayısı']].sum().reset_index()
    toplamaylarsort =toplamaylar.sort_values(by=['Vaka Sayısı'],ascending=False)
```

```

# Toplam Bolge , Cins grafikleri column sayısını belirtip with ile yazıyoruz
toplambc, toplamb, toplamc = st.columns(3)

with toplambc:
    st.markdown("Bölgelerde ortalamanın üzerinde saldırında bulunan cinsler")
    fig3 = px.treemap(data_frame=toplambolcinsf,
                        path=[px.Constant(''), 'Bolge', 'Cins'],
                        values='Vaka Sayisi',
                        width=900,
                        height=500)
    st.write(fig3)

with toplamb:
    st.markdown("Bölgelere göre toplam köpek saldıruları")
    fig = go.Figure(data=[go.Pie(labels=bolgelabel, values=bolgevalues, hole=.3
                                    )
                           ])
    fig.update_layout(margin=dict(t=5, b=4, l=4, r=4))
    st.write(fig)

with toplamc:
    st.markdown("Cinslere göre toplam köpek saldıruları ilk 10")
    fig2 = px.bar(
        data_frame=toplamcinsdf, x="Cins", y="Vaka Sayisi",
        barmode='group',
        width=900,
        height=500
    )
    st.write(fig2)

```

```

### dataframeleri detay tablo olarak yazdıracağım ancak burada df yazdırduğum zaman satır numaraları gözükmemesi
# için şu kodu uyguluyorum
# CSS to inject contained in a string
hide_table_row_index = """
<style>
thead tr th:first-child {display:none}
tbody th {display:none}
</style>
"""

# Inject CSS with Markdown
st.markdown(hide_table_row_index, unsafe_allow_html=True)

### DETAY GÖRÜNMÜLƏR adı altında yukarıda oluşturduğum df'leri tablo olarak yazdırıldım

st.title("Detay Görünlər")
st.markdown("")
aylartoplasm, toplambdetay, toplamcinsdetay = st.columns(3)

with aylartoplasm:
    st.markdown("Aylara Göre Toplam Saldırular")
    st.table(toplamaylarsort)
with toplambdetay:
    st.markdown("Bölgelere Göre Toplam Saldırular")
    st.table(toplambolgdef)
with toplamcinsdetay:
    st.markdown("Cinslere Göre Toplam Saldırular")
    st.table(toplamcinsdf)

time.sleep(1)

#scripti otomatik olarak yenilemek için cache silerek rerun yapıyorum
def clearcache():
    st.experimental_memo.clear()
clearcache()
time.sleep(2)
def rerun():
    st.experimental_rerun()
rerun()

```

STREAMLIT



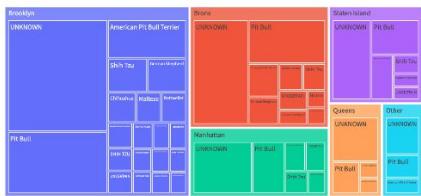
VISUALIZATION FINAL

NYC Köpek Saldırıları

Bölgelere ve cinslerine göre köpek saldıruları durumu

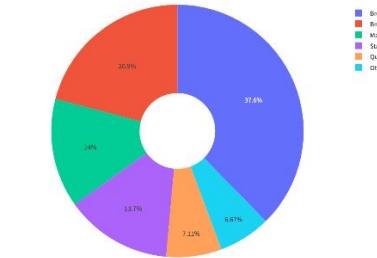
Bölge Sayısı
6

Bölgelerde ortalamadan üzerinde saldırında bulunan cinsler



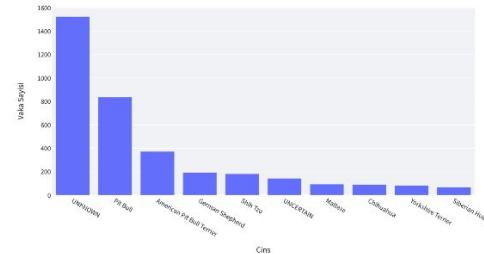
Cins Sayısı
167

Bölgelere göre toplam köpek saldıruları



Toplam Vaka
5440

Cinslere göre toplam köpek saldıruları ilk 10



Detay Görünlümler

Aylara Göre Toplam Saldırılar

Aylar	Vaka Sayısı
Ocak	801
Mayıs	746
Nisan	737
Subat	624
Mart	611
Haziran	580
Temmuz	404
Agustos	261
Eyluk	224
Ekim	197
Kasım	132
Aralık	123

Bölgelere Göre Toplam Saldırılar

Bölge	Vaka Sayısı
Brooklyn	2048
Bronx	1138
Manhattan	760
Staten Island	744
Queens	387
Other	363

Cinslere Göre Toplam Saldırılar

Cins	Vaka Sayısı
UNKNOWN	2048
Pit Bull	1138
American Pit Bull Terrier	760
German Shepherd	744
Shih Tzu	387
UNCERTAIN	363
Maltese	94
Chihuahua	90
Yorkshire Terrier	82
Siberian Husky	67

STREAMLIT



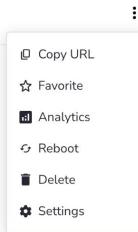
DEPLOYMENT

STREAMLIT üzerinde **GITHUB** ayarlarımı yaptım. **STREAMLIT CLOUD** üzerinde de **VM** üzerinde **DOCKER** ile de **DEPLOY** ederek kullanabiliriz bunun için uygulayacağım adımları bir sonraki sayfada anlatacağım.

keremngy's apps

New app ▾

streamlit · main · main.py



STREAMLIT KAYNAKLAR

<https://docs.streamlit.io/knowledge-base/tutorials/databases/bigquery>

<https://surendraredd.github.io/Books/examples.html>

<https://blog.streamlit.io/how-to-build-a-real-time-live-dashboard-with-streamlit/>

DOCKER



DEPLOY

STREAMLIT üzerinde oluşturduğum uygulamayı **CLOUD** üzerinde bir **VM** oluşturup **DOCKER** ile **DEPLOY** etmek istiyorum. Öncelikle **HTTP** ve **HTTPS** trafiklere izin veren bir **VM INSTANCE** oluşturup **DOCKER** yükleyeceğim. Ardından hazırladığım projeyi **DOCKERFILE** ile **IMAGE** olarak yapıp bu **IMAGE**'dan bir **CONTAINER** oluşturacağım. **DOCKERFILE** dosyasında **CREDENTIALS** dosyasını göndermek için **STREAMLIT** klasörünü oluşturup **SECRETS.TOML** dosyasını gönderiyorum. Aynı zamanda **MAIN.PY** dosyasınıda ekliyorum.

Firewall

Add tags and firewall rules to allow specific network traffic from the Internet

- Allow HTTP traffic
- Allow HTTPS traffic

VM INSTANCE içerisinde **SHELL** ile aşağıdaki komutları çalıştırıp **DOCKER** yüklemesi yaptım

```
sudo apt update
sudo apt install --yes apt-transport-https ca-certificates curl gnupg2 software-properties-common
curl -fsSL https://download.docker.com/linux/debian/gpg | sudo apt-key add -
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/debian ${lsb_release -cs} stable"
sudo apt update
sudo apt install --yes docker-ce
```

DOCKER komutlarını her seferinde **SUDO** ile yazmamak için şu komutu çalıştırıyorum

```
sudo usermod -aG docker $USER
```

```
# app/Dockerfile
FROM python:3.9-slim
EXPOSE 8501
WORKDIR /app

RUN apt-get update && apt-get install -y \
    build-essential \
    software-properties-common \
    git \
&& rm -rf /var/lib/apt/lists/*

RUN mkdir -p /app/.streamlit
ADD secrets.toml /app/.streamlit/secrets.toml
ADD main.py /app/main.py

# copy over and install packages
COPY requirements.txt ./requirements.txt
RUN pip3 install -r requirements.txt

# run app
ENTRYPOINT ["streamlit", "run", "main.py", "--server.port=8501",
"--server.address=0.0.0.0"]
```

DOCKER



DEPLOY FINAL

DOCKER ile **CONTAINER**'ı ayağa kaldırmadan önce bir **FIREWALL RULE** oluşturup **VM**'in **FIREWALL** ayarlarında **8501 PORT** için oluşturduğum **FIREWALL RULE TAG**'ını tanımladım. **DOCKER CONTAINER** 'ı **RUN** ederken **VM INSTANCE**'ın **INTERNAL IP**'si ile **RUN** ediyorum ve kısa bir süre sonra **EXTERNAL IP** üzerinden **8501 PORT**'u ile ulaşılabilir hale gelmiş oluyor. **VM EXTERNAL IP** ile proje bağlantıya hazır hale geliyor.

The screenshot shows the Firewall configuration interface. A search bar at the top right contains the text "streamlit". Below it, a table lists a single rule:

Name	Type	Targets	Filters	Protocols / ports	Action	Priority	Network	Logs	Hit count	Last hit
streamlit	Ingress	streamlit	IP ranges: 0.0.0.0/0	tcp:8501 udp:8501	Allow	1000	default	Off	—	—

To the right of the table, there are sections for "Firewalls" and "Network tags". Under "Firewalls", two rules are listed: "HTTP traffic" and "HTTPS traffic", both set to "On". Under "Network tags", three tags are listed: "http-server", "https-server", and "streamlit" (which is highlighted with a red arrow).

Dockerda dockerfile'dan bir image yaratıyoruz

docker build -t streamlit .

Dockerdaki imagedan bir container oluşturuyoruz buradaki -p container portunu server portuna publish ediyor

sudo docker run -d -p internalip:8501:8501 --name streamlit -h strmlt streamlit

```
:-$ sudo docker run -d -p 1    0.9:8501:8501 --name streamlit -h strmlt streamlit
34bc0adc8f2575cf77eda6aaaf8910e8b5676b3975b5ea171b14945938b5ead2
:-$ sudo docker image ls
REPOSITORY  TAG      IMAGE ID      CREATED      SIZE
streamlit   latest   e3eb45e2d887  8 minutes ago  1.18GB
python      3.9-slim f550e60adaa9  7 hours ago   125MB
:-$ sudo docker ps -a
CONTAINER ID  IMAGE      COMMAND      CREATED      STATUS      PORTS      NAMES
34bc0adc8f25  streamlit   "streamlit run main..."  About a minute ago  Up About a minute  1    0.9:8501->8501/tcp  streamlit
```



TEŞEKKÜRLER

BİR SONRAKİ PROJEDE
GÖRÜŞMEK ÜZERE