BTS | Barcelona Technology School

Real Time Data Analysis

Master in Big Data Solutions 2020-2021

Ankit Tewari
Course Instructor, Real Time Data Analysis (2020-21)
ankit.tewari@bts.tech

Today's Agenda

What are we going to discuss today?

- Understand the concept of twitter data streams
- Perform basic operations using twitter data streams
- Discuss major challenges and propose solutions while processing data streams
- Explore further production grade suggestions for stream processing applications

Contents

Today's topics at a glance

1. Introduction to Twitter's developer portal
2. Learning to stream twitter: Tweepy
3. Stream processing and aggregations
4. Tweepy's default streaming solution
5. Improving the application: Design Choices
6. Improving the application: Caching vs Persistence
7. Improving the application: Debugging
8. Improving the application: Model Management

Contents

Today's topics at a glance

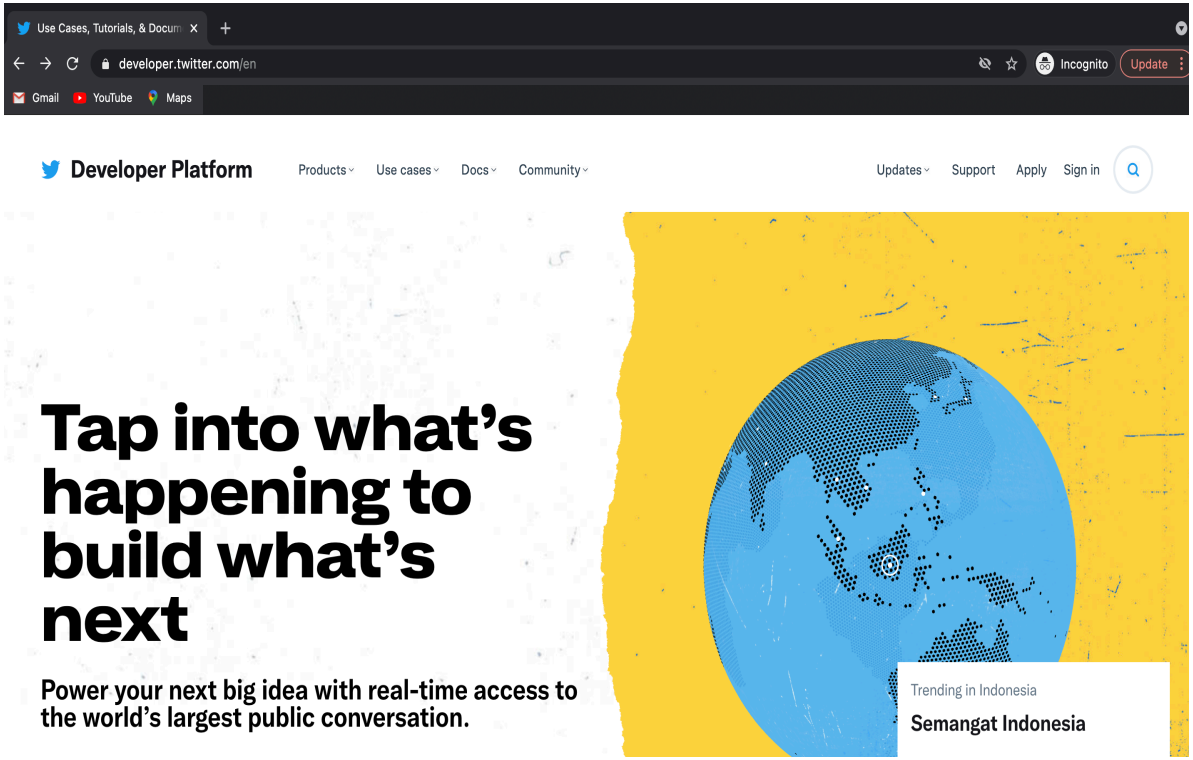
1. Introduction to Data Streams
2. Comparison between Data Batches and Data Streams
3. Stream-Processing Use-Cases
4. Advantages and Disadvantages of Stream Processing
5. Transformations on Data Streams
6. Input and Output
7. Triggers
8. Event Time and Stateful Processing
9. Production Grade Structured Streaming

Streaming Twitter

Introduction to developer portal

Twitter provides a developer portal that acts as a gateway to all the types of data provided by twitter. The portal also allows users to create their unique credentials and exploit the API.

1. In order to log in to the developer portal, log on to https://developer.twitter.com/en

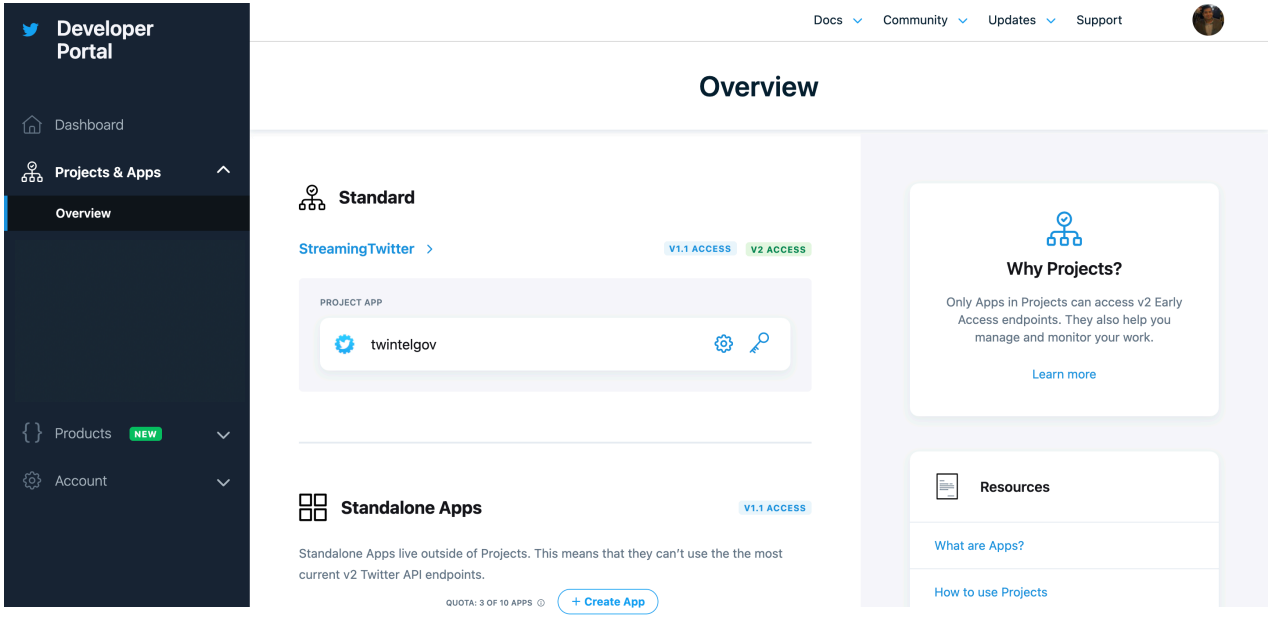


Streaming Twitter

Introduction to developer portal

Twitter provides a developer portal that acts as a gateway to all the types of data provided by twitter. The portal also allows users to create their unique credentials and exploit the API.

1. In order to log in to the developer portal, log on to https://developer.twitter.com/en
2. Once the login is successful, scroll down the page to find “Create App” feature

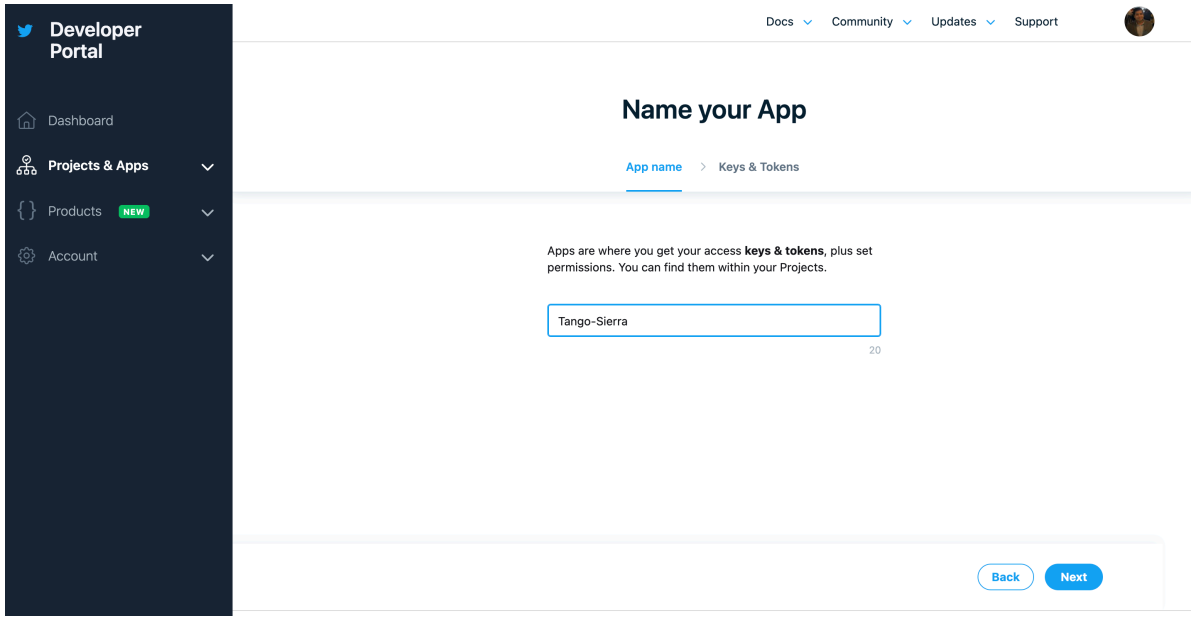


Streaming Twitter

Introduction to developer portal

Twitter provides a developer portal that acts as a gateway to all the types of data provided by twitter. The portal also allows users to create their unique credentials and exploit the API.

1. In order to log in to the developer portal, log on to https://developer.twitter.com/en
2. Once the login is successful, scroll down the page to find “Create App” feature



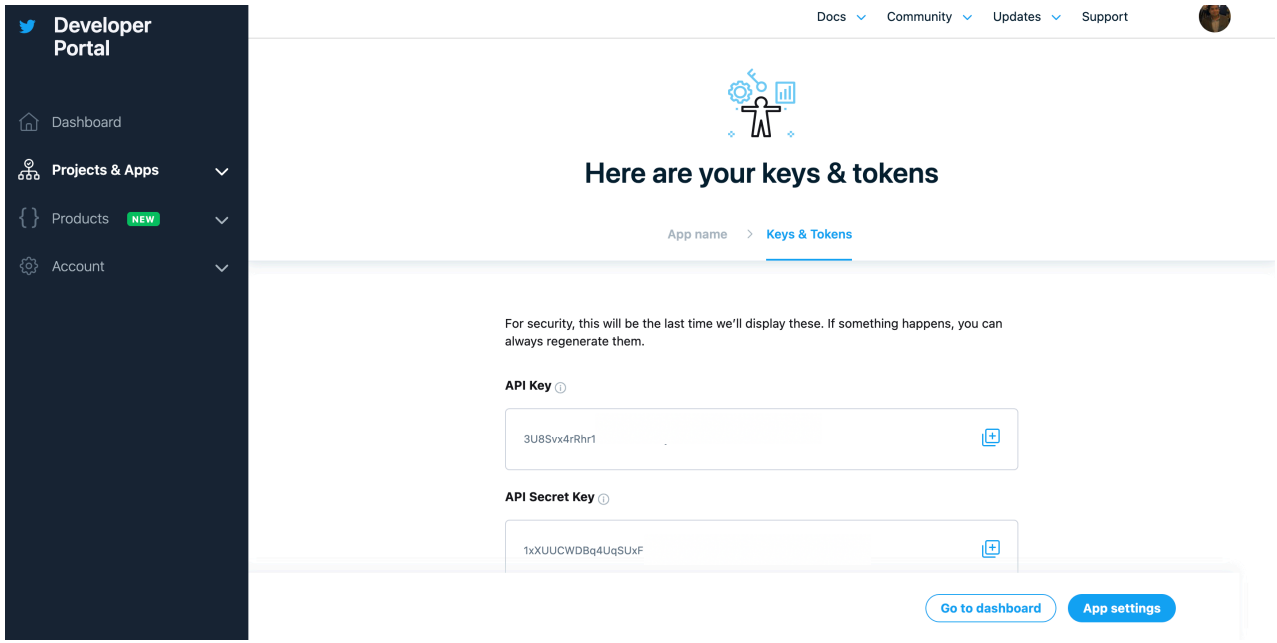
3. After clicking the ”Create App” button, we can see a page requesting a name of the application. Here, don’t forget to name it after the purpose of the application and not something random as we’ve provided for demonstration.

Streaming Twitter

Introduction to developer portal

Twitter provides a developer portal that acts as a gateway to all the types of data provided by twitter. The portal also allows users to create their unique credentials and exploit the API.

1. In order to log in to the developer portal, log on to https://developer.twitter.com/en
2. Once the login is successful, scroll down the page to find “Create App” feature



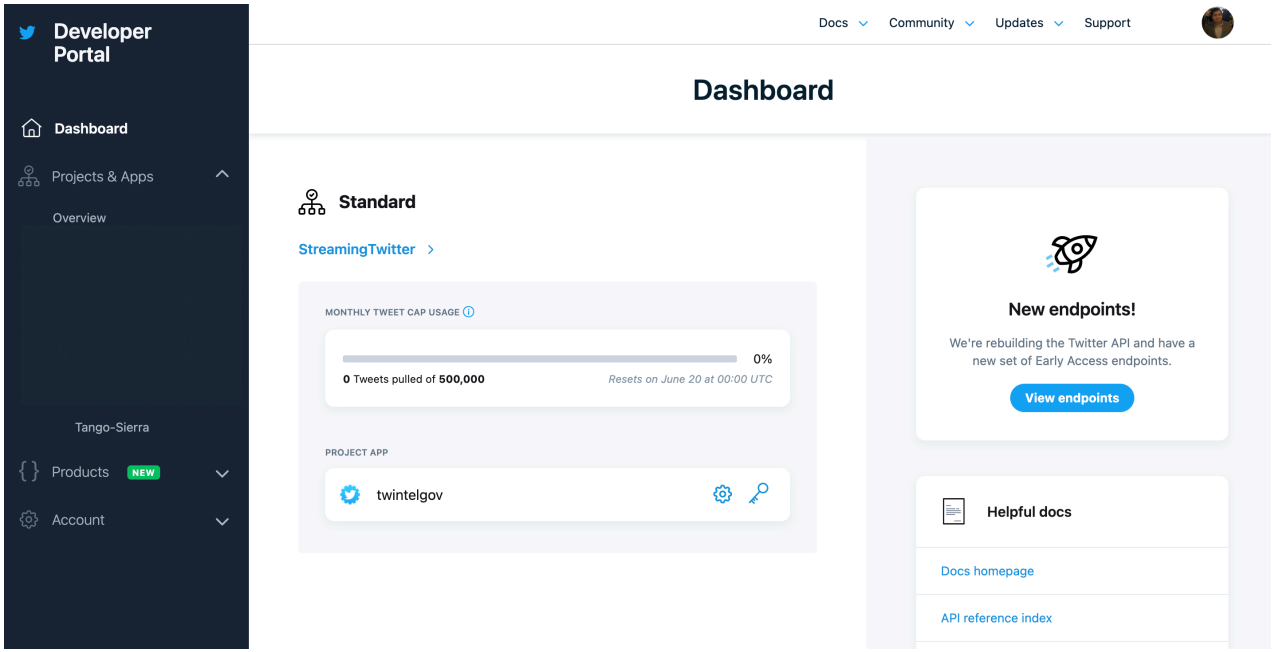
3. After clicking the ”Create App” button, we can see a page requesting a name of the application. Here, don’t forget to name it after the purpose of the application and not something random as we’ve provided for demonstration.
4. Thereafter, we can see our API Key and API Secret Key. These are one of the four authentication credentials. Remember, you’ve to write it down by copying and pasting because it may not be accessible again.
5. Now, we can go the dashboard back and check the name of our application to see if its created.

Streaming Twitter

Introduction to developer portal

Twitter provides a developer portal that acts as a gateway to all the types of data provided by twitter. The portal also allows users to create their unique credentials and exploit the API.

1. Now, once the application is created, we can click the “Product and Apps” drop-down menu and select the name of the application that we have created. In this case, it is “Tango-Sierra”.

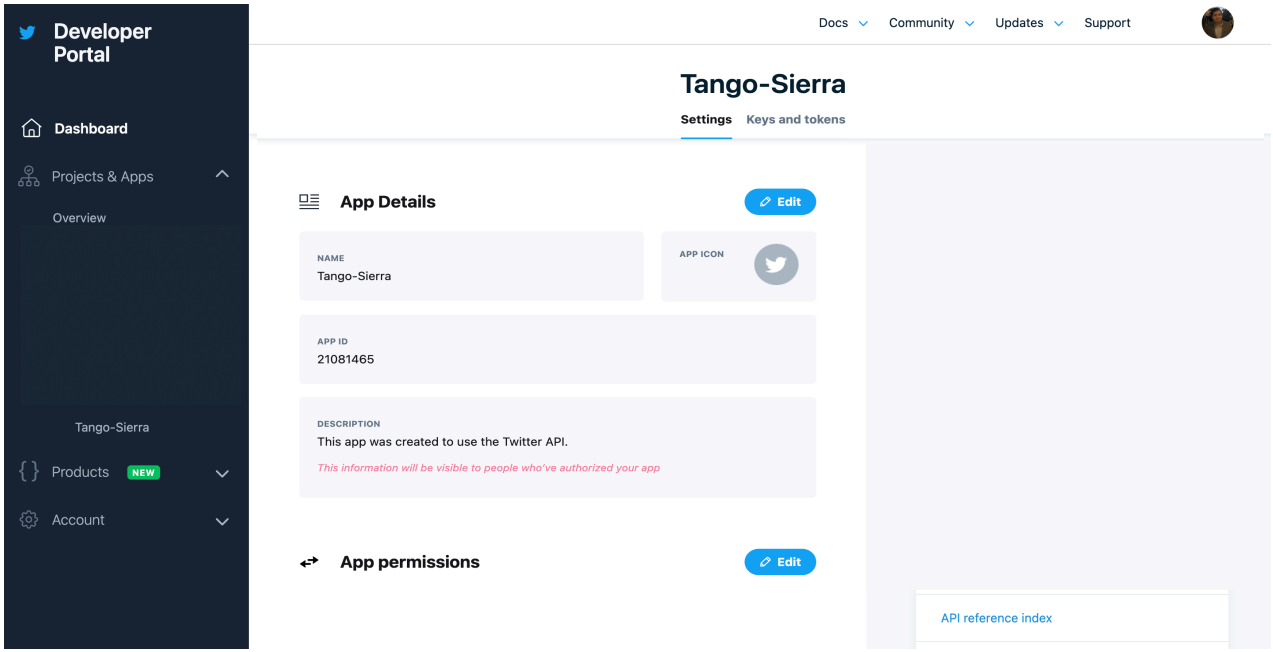


Streaming Twitter

Introduction to developer portal

Twitter provides a developer portal that acts as a gateway to all the types of data provided by twitter. The portal also allows users to create their unique credentials and exploit the API.

1. Now, once the application is created, we can click the “Product and Apps” drop-down menu and select the name of the application that we have created. In this case, “Tango-Sierra”.



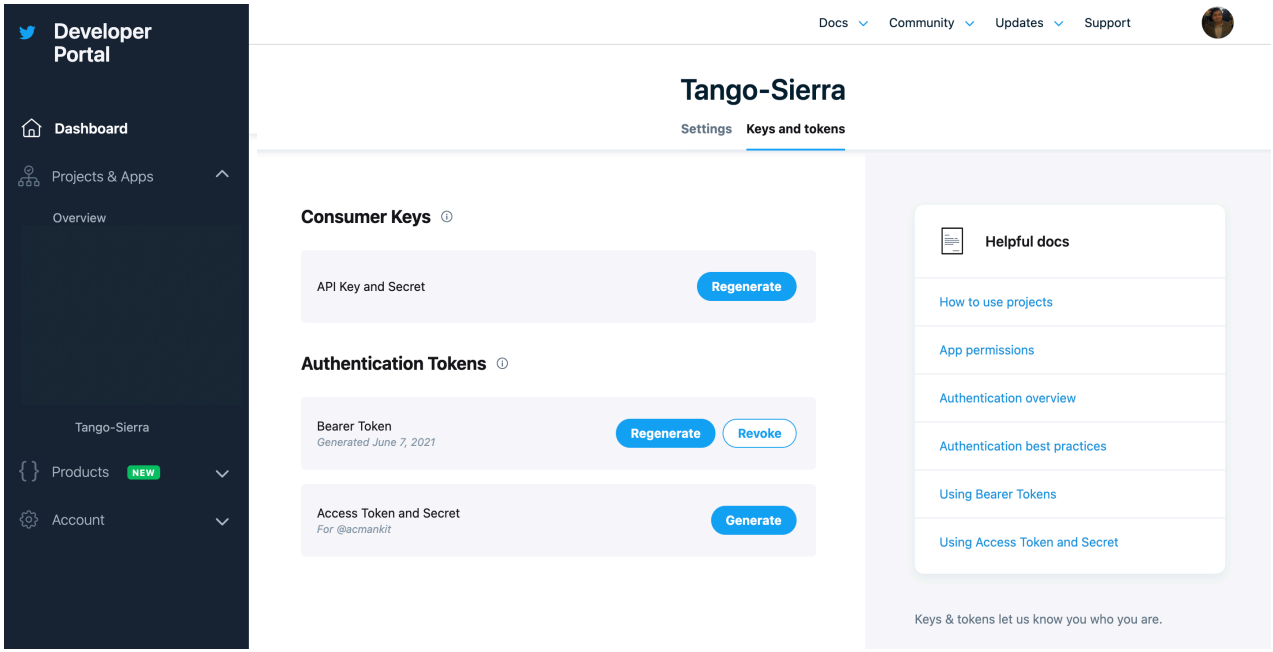
2. We can observe that we have two tabs- “Settings” and “Keys and tokens”. We need to switch to “Keys and tokens” in order to generate the tokens needed for this session.

Streaming Twitter

Introduction to developer portal

Twitter provides a developer portal that acts as a gateway to all the types of data provided by twitter. The portal also allows users to create their unique credentials and exploit the API.

1. Now, once the application is created, we can click the “Product and Apps” drop-down menu and select the name of the application that we have created. In this case, “Tango-Sierra”.

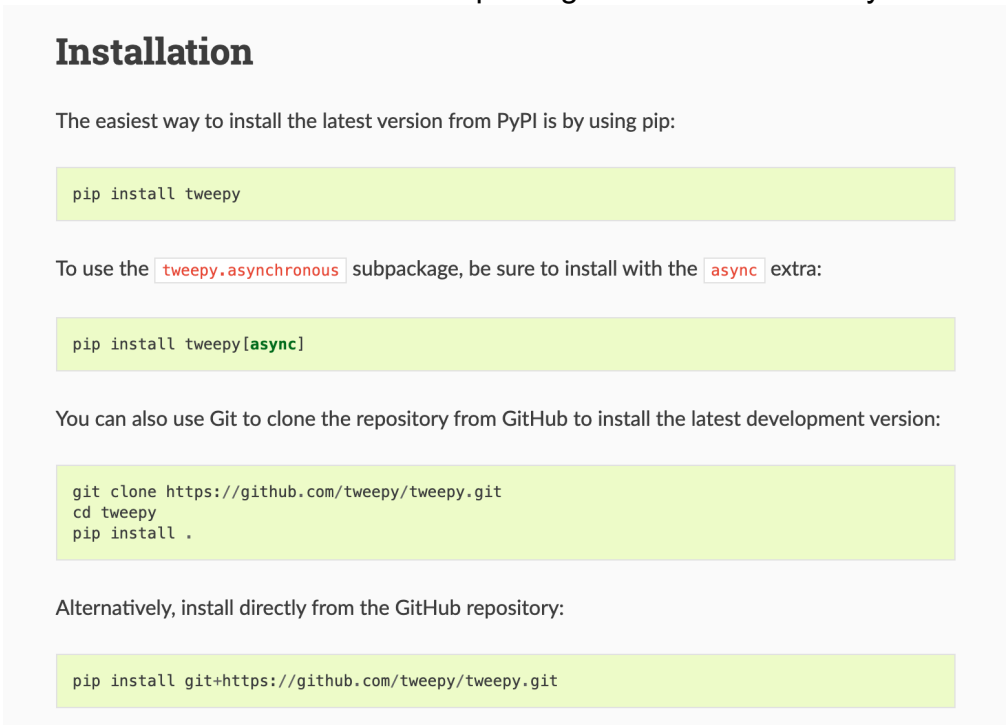


2. We can observe that we have two tabs- “Settings” and “Keys and tokens”. We need to switch to “Keys and tokens” in order to generate the tokens needed for this session.
3. Once, tokens are generated, they need to be saved because they’ll be used while authenticating the application.

Streaming Twitter

Introduction to Tweepy

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.
The first steps would be to start with the installation of the package. We can use the PyPI for this purpose or even Github directly.



Installation

The easiest way to install the latest version from PyPI is by using pip:

```
pip install tweepy
```

To use the `tweepy.asynchronous` subpackage, be sure to install with the `async` extra:

```
pip install tweepy[async]
```

You can also use Git to clone the repository from GitHub to install the latest development version:

```
git clone https://github.com/tweepy/tweepy.git
cd tweepy
pip install .
```

Alternatively, install directly from the GitHub repository:

```
pip install git+https://github.com/tweepy/tweepy.git
```

Streaming Twitter

Introduction to Tweepy

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Let us create our first code to authenticate the credentials and logging into the Twitter application that we have created.

```
import tweepy
import json
from pathlib import Path

ACCESS_TOKEN = "1288431932
ACCESS_SECRET =  "krPnYxZg
CONSUMER_KEY =  "1Lj8DS1pW
CONSUMER_SECRET =  "DVtoIg

auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)

api = tweepy.API(auth)
```

Streaming Twitter

Introduction to Tweepy

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

```
public_tweets = api.home_timeline()
for tweet in public_tweets:
    print(tweet.text)
```

the object "public_tweets" behaves like a list in terms of accessing its elements

```
#COVID19 | Delhi will begin unlocking further from today as more relaxations will come into effect on the day… http
s://t.co/7cK0qCUmgw
Man Allegedly Pushes Wife, Daughters Into Well, 8-Year-Old Killed: Police https://t.co/s7mvJDWqt1 https://t.co/tZBJ
xLa7E0
RT @BrigMahalingam: Is Gaza blockade lifting?
RT @BrigMahalingam: Egypt mediation: Head of Hamas in Gaza says, he expects major breakthrough in humanitarian &amp
; economic condition of Gaza…
#COVID19 vaccine: Special 'pink booths' to start functioning from today for women in Uttar Pradesh… https://t.co/vx
zDJltvzV
#AIRExclusive interview with Chief Election Commissioner Sushil Chandra :

Part-1:https://t.co/qQVunkN5nI

Part-2 : https://t.co/nEDq0UObFb
Following the Galwan clash, the People's Liberation Army (#PLA) appears to be giving a fair share of importance to…
https://t.co/1MBUGXcNnb
With a little over 100,000 new infections, India logs lowest daily #COVID19 cases in 2 months: Data
```

Streaming Twitter

Introduction to Tweepy

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

```python
# Extract the information for a specific user
user = api.get_user('USAndIndia')
print(user.screen_name)
print(user.followers_count)
for friend in user.friends():
    print(friend.screen_name)
```

```
USAndIndia
209399
SandhuTaranjitS
SecBecerra
AmbassadorTai
PowerUSAID
Surgeon_General
DeputySecState
PressSec
WHNSC
USAmbUN
JakeSullivan46
ClimateEnvoy
SecDef
POTUS
VP
SecBlinken
WhiteHouse
USEmbassyFrance
AkashvaniAIR
DDIndialive
videovolunteers
```

This account corresponds to the Ambassador of India to United States. It of course makes sense that these account will be friends ;)

Streaming Twitter

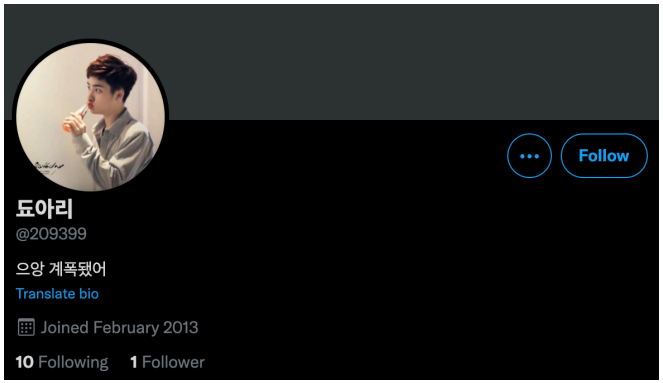
Introduction to Tweepy

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

```python
# Extract the information for a specific user
user = api.get_user('USAndIndia')
print(user.screen_name)
print(user.followers_count)
for friend in user.friends():
    print(friend.screen_name)
```

```
USAndIndia
209399
SandhuTaranjitS
SecBecerra
AmbassadorTai
PowerUSAID
Surgeon_General
DeputySecState
PressSec
WHNSC
USAmbUN
JakeSullivan46
ClimateEnvoy
SecDef
POTUS
VP
SecBlinken
WhiteHouse
USEmbassyFrance
AkashvaniAIR
DDIndialive
videovolunteers
```

This account appears strange. Let us dig a little deeper ;)



Streaming Twitter

Introduction to Tweepy

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

```python
# Extract the information for a specific user
user = api.get_user('USAndIndia')
print(user.screen_name)
print(user.followers_count)
for friend in user.friends():
    print(friend.screen_name)
```

```
USAndIndia
209399
SandhuTaranjitS
SecBecerra
AmbassadorTai
PowerUSAID
Surgeon_General
DeputySecState
PressSec
WHNSC
USAmbUN
JakeSullivan46
ClimateEnvoy
SecDef
POTUS
VP
SecBlinken
WhiteHouse
USEmbassyFrance
AkashvaniAIR
DDIndialive
videovolunteers
```

Actually, it's not suspicious at all in the context of US Embassy, India. We forgot that it prints the total number of followers that the US Embassy has :P

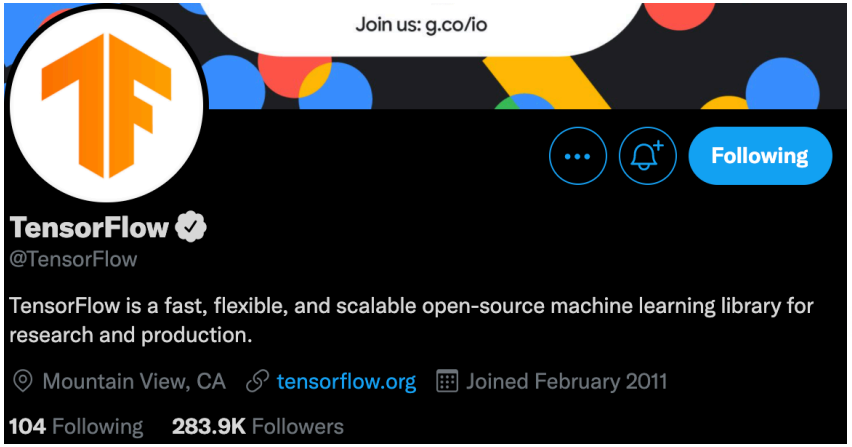
Streaming Twitter

Introduction to Tweepy

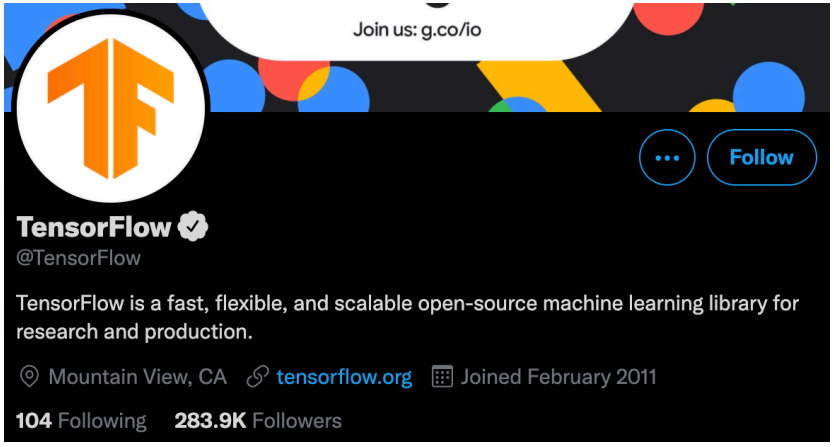
Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Let us explore if we can follow and unfollow the page of "TensorFlow" using our Tweepy package.There are many more things we can do with this package. See the notebook and documentation for more.

```
# to follow a given profile
api.create_friendship("TensorFlow")
```




```
# to follow or unfollow a given profile
api.destroy_friendship("TensorFlow")
```



Streaming Twitter

Getting started with directory for streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Let us start with first creating a sub-folder which can download json files about tweets for us from time to time once we make the request

```
import json
from pathlib import Path

public_tweets = api.home_timeline()
for tweet in public_tweets:
    base= Path("twitter-logs")
    jsonpath = base / (tweet.id_str + ".json")
    base.mkdir(exist_ok=True)
    jsonpath.write_text(json.dumps(tweet._json))
```

Streaming Twitter

Spark Structured Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Next, let's start with our structured streaming object. Observe carefully how we are specifying the schema.

```
IN_PATH = "./twitter-logs/"
OUT_PATH = ""
timestampformat = "EEE MMM dd HH:mm:ss zzzz yyyy"

spark = SparkSession.builder.appName("Twitter-Streaming").getOrCreate()
static= spark.read.json(IN_PATH)
schema = static.limit(100).schema
streaming = spark.readStream.schema(schema).option("maxFilesPerTrigger", 10)\
.json("./twitter-logs/")
```

Streaming Twitter

Spark Structured Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Now, let's explore a sub portion of the schema. Be careful not to be confused with the fact that the image below provides complete schema. In fact, it provides only a subset for the sake of convenience. Refer to notebook for more information.

```
static.printSchema()
```

```
root
 |-- contributors: string (nullable = true)
 |-- coordinates: string (nullable = true)
 |-- created_at: string (nullable = true)
 |-- entities: struct (nullable = true)
 |    |-- hashtags: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- indices: array (nullable = true)
 |    |    |    |    |-- element: long (containsNull = true)
 |    |    |    |-- text: string (nullable = true)
 |    |-- media: array (nullable = true)
 |    |    |-- element: struct (containsNull = true)
 |    |    |    |-- display_url: string (nullable = true)
 |    |    |    |-- expanded_url: string (nullable = true)
 |    |    |    |-- id: long (nullable = true)
 |    |    |    |-- id_str: string (nullable = true)
 |    |    |    |-- indices: array (nullable = true)
 |    |    |    |    |-- element: long (containsNull = true)
 |    |    |    |-- media_url: string (nullable = true)
```

Streaming Twitter

Spark Structured Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Now, it's the time for making some basic aggregation. Here, we are simply trying to group users by the count of their tweets.

```
# examining various counts of tweets grouped by user
tweet_counts = streaming.groupBy("user").count()

tweet_query = tweet_counts.writeStream.queryName("tweet_counts")\
.format("memory").outputMode("complete")\
.start()

spark.sql("SELECT * FROM tweet_counts").show()
```

```
+--------------------+-----+
|                user|count|
+--------------------+-----+
|{false, Sun Apr 0...|    1|
|{false, Sat Oct 3...|    1|
|{false, Mon Mar 0...|    1|
|{false, Tue Feb 1...|    1|
|{false, Wed Apr 2...|    2|
|{false, Fri Feb 1...|    1|
|{false, Fri Aug 2...|    3|
+--------------------+-----+
```

Streaming Twitter

Spark Structured Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.
It was interesting to do some aggregation. However, let's focus on a broader set of metrics for each user and refrain to selection.

```
selection_query = streaming\
.select("user", "user.favourites_count", "user.followers_count",
        "user.friends_count", "user.statuses_count")\
.writeStream\
.queryName("tweet_metrics")\
.format("memory")\
.outputMode("append")\
.start()
```

```
spark.sql("SELECT * FROM tweet_metrics").show()
```

```
+--------------------+----------------+---------------+-------------+---------------+
|                user|favourites_count|followers_count|friends_count|statuses_count|
+--------------------+----------------+---------------+-------------+---------------+
|{false, Sat Oct 3...|             219|         338337|          305|          38742|
|{false, Tue Feb 1...|             525|         804194|         2006|          78242|
|{false, Sun Apr 0...|            1164|       18835049|         1036|         341412|
|{false, Wed Apr 2...|            2929|        8084424|          132|         899825|
|{false, Mon Mar 0...|            3998|         212584|         1389|          69542|
|{false, Wed Apr 2...|            2929|        8084424|          132|         899825|
|{false, Fri Aug 2...|              88|          94766|          130|         198564|
|{false, Fri Aug 2...|              88|          94766|          130|         198564|
|{false, Fri Aug 2...|              88|          94766|          130|         198564|
|{false, Fri Feb 1...|             153|        6979140|          211|         379819|
|{false, Tue Aug 2...|            7228|         243128|          660|          16091|
|{false, Fri May 0...|               0|       15645033|           15|         818754|
|{false, Sun Apr 0...|            1164|       18835049|         1036|         341412|
|{false, Wed May 1...|             409|        3952143|           44|         516645|
|{false, Mon Feb 0...|            4445|        5705881|          248|         880101|
|{false, Thu Jan 2...|               4|       10063639|          377|         675016|
|{false, Thu May 1...|            1404|         180117|          844|          90388|
|{false, Fri Feb 1...|             153|        6979140|          211|         379819|
|{false, Fri Aug 2...|              88|          94766|          130|         198564|
|{false, Fri Aug 2...|              88|          94766|          130|         198564|
+--------------------+----------------+---------------+-------------+---------------+
only showing top 20 rows
```

Streaming Twitter

Spark Structured Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.
Now, let's refine our previous query by getting only user id for each of the selections for a more neat display.

```
selection_query = streaming\
.select("user.id", "user.favourites_count", "user.followers_count",
        "user.friends_count", "user.statuses_count")\
.writeStream\
.queryName("tweet_metrics_agg")\
.format("memory")\
.outputMode("append")\
.start()
```

```
spark.sql("SELECT * FROM tweet_metrics_agg").show(truncate= False)
```

```
+---------+----------------+---------------+-------------+---------------+
|id       |favourites_count|followers_count|friends_count|statuses_count|
+---------+----------------+---------------+-------------+---------------+
|86478153 |219             |338337         |305          |38742          |
|21114659 |525             |804194         |2006         |78242          |
|3108351  |1164            |18835049       |1036         |341412         |
|36327407 |2929            |8084424        |132          |899825         |
|22545453 |3998            |212584         |1389         |69542          |
|36327407 |2929            |8084424        |132          |899825         |
|180748385|88              |94766          |130          |198564         |
|180748385|88              |94766          |130          |198564         |
|180748385|88              |94766          |130          |198564         |
|20751449 |153             |6979140        |211          |379819         |
|68746721 |7228            |243128         |660          |16091          |
|37034483 |0               |15645033       |15           |818754         |
|3108351  |1164            |18835049       |1036         |341412         |
|39743812 |409             |3952143        |44           |516645         |
|19897138 |4445            |5705881        |248          |880101         |
|240649814|4               |10063639       |377          |675016         |
|39922594 |1404            |180117         |844          |90388          |
|20751449 |153             |6979140        |211          |379819         |
|180748385|88              |94766          |130          |198564         |
|180748385|88              |94766          |130          |198564         |
+---------+----------------+---------------+-------------+---------------+
only showing top 20 rows
```

Streaming Twitter

Spark Structured Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.
Going further, let us try with selecting retweet counts for each of the users. Observe we may have duplicates.

```
selection_query = streaming\
.select("user", "retweet_count", "lang", "created_at")\
.writeStream\
.queryName("selection_query_")\
.format("memory")\
.outputMode("append")\
.start()
```

```
spark.sql("SELECT * FROM selection_query_").show()
```

```
+--------------------+-------------+----+--------------------+
|                user|retweet_count|lang|          created_at|
+--------------------+-------------+----+--------------------+
|{false, Sat Oct 3...|            0|  en|Sun Jun 06 23:50:...|
|{false, Tue Feb 1...|            5|  en|Sun Jun 06 23:39:...|
|{false, Sun Apr 0...|            8|  en|Sun Jun 06 23:45:...|
|{false, Wed Apr 2...|            0|  en|Sun Jun 06 23:40:...|
|{false, Mon Mar 0...|            1|  en|Sun Jun 06 23:39:...|
|{false, Wed Apr 2...|            2|  en|Sun Jun 06 23:50:...|
|{false, Fri Aug 2...|            0|  en|Sun Jun 06 23:45:...|
|{false, Fri Aug 2...|            0|  en|Sun Jun 06 23:50:...|
|{false, Fri Aug 2...|            0|  en|Sun Jun 06 23:40:...|
|{false, Fri Feb 1...|            0|  en|Sun Jun 06 23:50:...|
|{false, Tue Aug 2...|            8|  en|Sun Jun 06 23:27:...|
|{false, Fri May 0...|            3|  en|Sun Jun 06 23:30:...|
|{false, Sun Apr 0...|           12|  en|Sun Jun 06 23:30:...|
|{false, Wed May 1...|            1|  en|Sun Jun 06 23:30:...|
|{false, Mon Feb 0...|            2|  en|Sun Jun 06 23:30:...|
|{false, Thu Jan 2...|            0|  en|Sun Jun 06 23:30:...|
|{false, Thu May 1...|            2|  en|Sun Jun 06 23:30:...|
|{false, Fri Feb 1...|            6|  en|Sun Jun 06 23:30:...|
|{false, Fri Aug 2...|            0|  en|Sun Jun 06 23:35:...|
|{false, Fri Aug 2...|            0|  en|Sun Jun 06 23:30:...|
+--------------------+-------------+----+--------------------+
only showing top 20 rows
```

Streaming Twitter

Spark Structured Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.
Finally, let's come back to our previous example and learn to filter duplicates using an aggregation query.

```
selection_query = streaming\
.select("user.id", "user.favourites_count", "user.followers_count",
        "user.friends_count", "user.statuses_count")\
.groupBy("id").count()
```

```
selection_query\
.writeStream\
.queryName("tweet_metrics_avg")\
.format("memory")\
.outputMode("complete")\
.start()
```

```
<pyspark.sql.streaming.StreamingQuery at 0x12fd2bad0>
```

```
spark.sql("SELECT * FROM tweet_metrics_avg").show(truncate= False)
```

```
+----------+-----+
|id        |count|
+----------+-----+
|22545453  |1    |
|39922594  |1    |
|37034483  |5    |
|43855487  |3    |
|134758540 |1    |
|22763833  |3    |
|240649814 |1    |
|68746721  |1    |
|3108351   |2    |
|1191912552|2    |
|86478153  |1    |
|180748385 |6    |
|36327407  |2    |
|19897138  |2    |
|21114659  |1    |
|39743812  |1    |
|920488039 |2    |
|20751449  |2    |
|38647512  |2    |
|48008938  |1    |
+----------+-----+
```

Streaming Twitter

Default Tweepy Streaming

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Here, we have an example of default streaming feature presented by Tweepy.

```python
import json
import tweepy

class MyStreamListener(tweepy.StreamListener):
    def __init__(self, api):
        self.api = api
        self.me = api.me()

    def on_status(self, tweet):
        print(f"{tweet.user.name}:{tweet.text}")

    def on_error(self, status):
        print("Error detected")

# Authenticate to Twitter
auth = tweepy.OAuthHandler(CONSUMER_KEY, CONSUMER_SECRET)
auth.set_access_token(ACCESS_TOKEN, ACCESS_SECRET)

# Create API object
api = tweepy.API(auth, wait_on_rate_limit=True,
    wait_on_rate_limit_notify=True)

tweets_listener = MyStreamListener(api)
stream = tweepy.Stream(api.auth, tweets_listener)
stream.filter(track=["Covid-19", "Trump", "Whatsapp"], languages=["en"])
```

Streaming Twitter

Introduction to developer portal

Tweepy is the python package which is used to access the Twitter's vast amount of social data. In this tutorial, we will refrain ourselves to a version 3.1.0. However, a latest release has already been made and docs are available for version 4.0.0.

Here, we have an example of default streaming feature presented by Tweepy.

```
The Health Site:COVID-19 Pandemic Sparks Boom in Online Fitness Training: Tips For Virtual Workout Classes
 #AtHomeExercises… https://t.co/u6qxHrrl28
Nepal News English:TKP: Rehabilitation scheme for Covid-19 affected enterprises fails to woo borrowers https://t.co
/rLKkxH707Q https://t.co/WVZx8lLZfk
Aliza Stone:Can obesity increase the risk of long-term COVID-19 complications, here is what study at Cleveland Clin
ic says
https://t.co/2xf2qm8jxt
Dom:RT @malaymail: NSC: No need to update MySejahtera profile unless showing Covid-19 signs, been a close contact o
r abroad https://t.co/2DIlol…
Mi India Support:@your_raghvendra Please be informed that as a measure of precaution against COVID-19 and owing to
restrictions, our… https://t.co/ZpeIbkEBS1
Jo-Anne Wiszniewski:RT @SAHealth: The Tailem Bend COVID-19 mobile testing clinic will close at 1.30pm today, Monday
7 June, due to strong winds and dust storms.
อาคม ย.:RT @armyspoke_news: 🇹🇭The 9th Infantry Division inspected the admission examination of the Armed Forces Aca
demies Preparatory School (RTA)…
Nepal News English:TKP: Nepal signs non-disclosure agreement to buy Chinese Covid-19 vaccines but legal questions r
emain… https://t.co/znPwfHzFXq
```

Indirect Application Performance Enhancements

Design Choices

In terms of design choices, following two questions arrive extremely frequently whenever we are developing a production grade Apache Spark application and not just for the sake of prototyping.

1. Scala vs Java vs Python vs R:
   In general, while it is extremely dependent from case-to-case. However, if we generalise such as performing an extract-transform-load operation and then running some single node machine learning, then SparkR would be most beneficial as it allows access to massive machine learning ecosystem of R.

   However, if we are doing some transformations that can not be done using structured APIs, such as RDD transformations then Python or R are probably not the best choice. In such cases, it is best advisable to that using Python for the majority of the application, and porting some of it to Scala;

2. DataFrames vs SQL vs Datasets vs RDD:
   In general, all these features are equivalent in speed across different languages. However, the actual performance hit is observed when we try do to something like writing User-Defined-Functions. In this case, we end up suffering in languages like Python or R versus Scala or Java

Direct Application Performance Enhancements

Data Caching

In applications that reuse the same datasets over and over, one of the most useful optimizations is caching. This technique will place a DataFrame, table, or RDD into temporary storage (either memory or disk) across the executors in your cluster, and make subsequent reads faster.

Although caching might sound like something we should do all the time, it's not always a good thing to do. That's because caching data incurs a serialization, deserialization, and storage cost. For example, if you are only going to process a dataset once (in a later transformation), caching it will only slow you down.

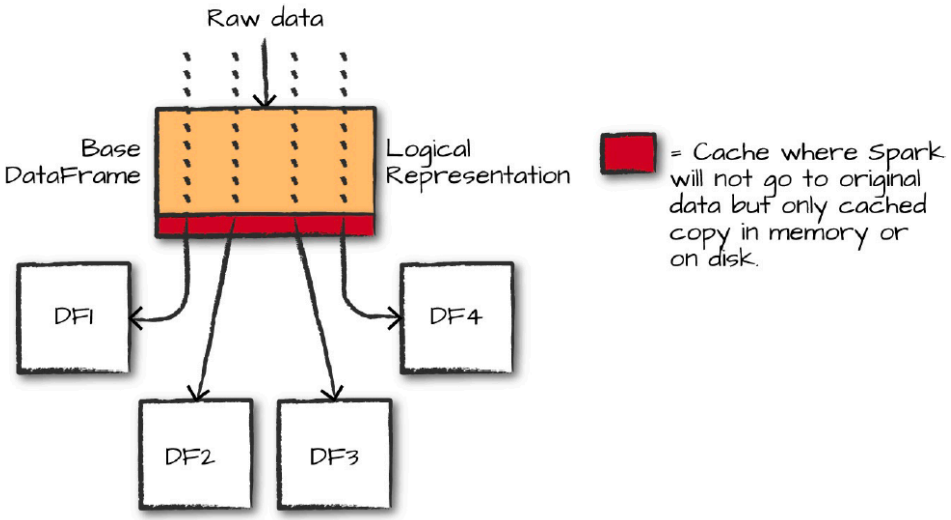
Caching is a lazy operation, meaning that things will be cached only as they are accessed. The RDD API and the Structured API differ in how they actually perform caching.

When we cache an RDD, we cache the actual, physical data (i.e., the bits). The bits. When this data is accessed again, Spark returns the proper data. This is done through the RDD reference. However, in the Structured API, caching is done based on the physical plan. This means that we effectively store the physical plan as our key (as opposed to the object reference) and perform a lookup prior to the execution of a Structured job.

Direct Application Performance Enhancements

Data Caching

In applications that reuse the same datasets over and over, one of the most useful optimizations is caching. This technique will place a DataFrame, table, or RDD into temporary storage (either memory or disk) across the executors in your cluster, and make subsequent reads faster.



We load an initial DataFrame from a CSV file and then derive some new DataFrames from it using transformations. We can avoid having to recompute the original DataFrame (i.e., load and parse the CSV file) many times by adding a line to cache it along the way.

Direct Application Performance Enhancements

Data Caching

In applications that reuse the same datasets over and over, one of the most useful optimizations is caching. This technique will place a DataFrame, table, or RDD into temporary storage (either memory or disk) across the executors in your cluster, and make subsequent reads faster.

Now, let us examine the performance by comparing (in terms of actual runtime of some aggregation code) when the DataFrame is read natively versus when it is cached.

```
start= time.time()
sales = spark.read.csv('online-retail-dataset.csv', header= True, inferSchema= True)
sales_country= sales.groupBy("Country").count().collect()
sales_customer= sales.groupBy("CustomerID").count().collect()
sales_invoice= sales.groupBy("InvoiceNo").count().collect()
end= time.time()
print("total time elapsed: ", end-start, "seconds")
```

```
total time elapsed:  9.267887830734253 seconds
```

```
sales.cache()
sales.count()
start= time.time()
sales_country= sales.groupBy("Country").count().collect()
sales_customer= sales.groupBy("CustomerID").count().collect()
sales_invoice= sales.groupBy("InvoiceNo").count().collect()
end= time.time()
print("total time elapsed: ", end-start, "seconds")
```

```
total time elapsed:  1.9063010215759277 seconds
```

And, you can see yourself that it is actually a performance increase by a factor of approximately,

4.88x

Direct Application Performance Enhancements

Data Persistence

Persistence of data is similar to caching but nuanced, providing control over how your data is cached via StorageLevel. Data on disk is always serialized using either Java or Kryo serialization. Here we can see some of the storage levels-

StorageLevel	Description
MEMORY_ONLY	Data is stored directly as objects and stored only in memory.
MEMORY_ONLY_SER	Data is serialized as compact byte array representation and stored only in memory. To use it, it has to be deserialized at a cost.
MEMORY_AND_DISK	Data is stored directly as objects in memory, but if there's insufficient memory the rest is serialized and stored on disk.
DISK_ONLY	Data is serialized and stored on disk.
OFF_HEAP	Data is stored off-heap. Off-heap memory is used in Spark for storage and query execution; see "Configuring Spark executors' memory and the shuffle service" on page 178.
MEMORY_AND_DISK_SER	Like MEMORY_AND_DISK, but data is serialized when stored in memory. (Data is always serialized when stored on disk.)

If no StorageLevel is specified then the default is MEMORY_AND_DISK_DESER;

Direct Application Performance Enhancements

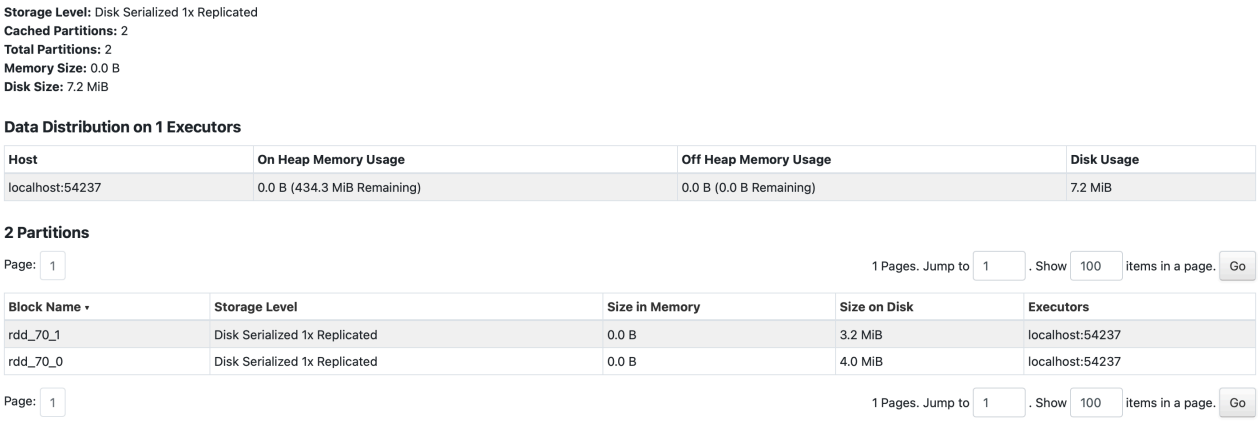
Data Persistence

Persistence of data is similar to caching but nuanced, providing control over how your data is cached via StorageLevel. Data on disk is always serialized using either Java or Kryo serialization. The code below explains how to persist the DataFrame-

```
from pyspark.storagelevel import StorageLevel
sales.persist(storageLevel= StorageLevel.DISK_ONLY)
sales.count()
```

```
541909
```

Eventually, we can verify the storage related information from the Spark-UI to confirm about the type of storage as well as number of partitions etc.



Storage Level: Disk Serialized 1x Replicated
Cached Partitions: 2
Total Partitions: 2
Memory Size: 0.0 B
Disk Size: 7.2 MiB

Data Distribution on 1 Executors

Host	On Heap Memory Usage	Off Heap Memory Usage	Disk Usage
localhost:54237	0.0 B (434.3 MiB Remaining)	0.0 B (0.0 B Remaining)	7.2 MiB

2 Partitions

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Block Name	Storage Level	Size in Memory	Size on Disk	Executors
rdd_70_1	Disk Serialized 1x Replicated	0.0 B	3.2 MiB	localhost:54237
rdd_70_0	Disk Serialized 1x Replicated	0.0 B	4.0 MiB	localhost:54237

Page: 1 — 1 Pages. Jump to 1 . Show 100 items in a page. Go

Debugging Application

Slow I/O Operations

Slow I/O can be difficult to diagnose, especially with networked file systems. Some of the signs of such an operation are-

- Slow reading of data from a distributed file system or external system.
- Slow writes from network file systems or blob storage.

In such cases, possible remedies maybe-

1. For distributed file systems such as HDFS running on the same nodes as Spark, make sure Spark sees the same hostnames for nodes as the file system. This will enable Spark to do locality-aware scheduling.
2. Turning on speculation (set spark.speculation to true) can help with slow reads and writes. This will launch additional tasks with the same operation in an attempt to see whether it's just some transient issue in the first task. Speculation is a powerful tool and works well with consistent file systems.
3. Finally, check for sufficient network connectivity because the Spark cluster may not have enough total network bandwidth to get to the storage system.

Debugging Application

Slow Aggregations

Slow aggregations can cause significant loss of time and costs specially in real time reporting systems. Some of the signs of such an operation are-

- Slow tasks during a groupBy call.
- Jobs after the aggregation are slow, as well.

In such cases, possible remedies maybe-

1. We may try increasing the number of partitions, prior to an aggregation, might help by reducing the number of different keys processed in each task.
2. We may also try increasing executor memory can help alleviate this issue, as well. If a single key has lots of data, this will allow its executor to spill to disk less often and finish faster, although it may still be much slower than executors processing other keys.
3. If we find that tasks after the aggregation are also slow, this means that our dataset might have remained unbalanced after the aggregation. We may try inserting a repartition call to partition it randomly.
4. Ensuring that all filters and SELECT statements that can be are above the aggregation can help to ensure that you're working only on the data that you need to be working on and nothing else. Spark's query optimizer will automatically do this for the structured APIs.
5. Ensure null values are represented correctly (using Spark's concept of null) and not as some default value like " " or "EMPTY". Spark often optimizes for skipping nulls early in the job when possible, but it can't do so for your own placeholder values.
6. Some aggregation functions are also just inherently slower than others. For instance, collect_list and collect_set are very slow aggregation functions because they must return all the matching objects to the driver, and should be avoided in performance-critical code.

Application Model Management

Understanding Reproducibility

Before we deploy our machine learning model, we should ensure that we can reproduce and track the model's performance.

Having end-to-end reproducibility of machine learning solutions means that:

- we need to be able to reproduce the code that generated a model,
- the environment used in training,
- the data it was trained on, and
- the model itself.

One approach to start with is to set your seeds so you can reproduce your experiments (e.g., for the train/test split, when using models with inherent randomness such as random forests).

Application Model Management

Setting seeds isn't enough!

Before we deploy our machine learning model, we should ensure that we can reproduce and track the model's performance.

Here are examples of model reproducibility-

1. Library Versioning
2. Data Evolution
3. Execution Sequence
4. Parallel Operations

Application Model Management

Setting seeds isn't enough!

Before we deploy our machine learning model, we should ensure that we can reproduce and track the model's performance.

Here are examples of model reproducibility-

1. Library Versioning:
   When a data scientist hands over his code, it may or may not mention the dependent libraries.

   While we are able to figure out which libraries are required by going through the error messages, we won't be certain which library versions they used, so we are likely install the latest ones.

   But if their code was built on a previous version of a library, which may be taking advantage of some default behaviour that differs from the version we installed, using the latest version can cause the code to break or the results to differ (for example, consider how XGBoost changed how it handles missing values in v0.90).

2. Data Evolution
3. Execution Sequence
4. Parallel Operations

Application Model Management

Setting seeds isn't enough!

Before we deploy our machine learning model, we should ensure that we can reproduce and track the model's performance.

Here are examples of model reproducibility-

1. Library Versioning
2. Data Evolution:
   Suppose we build a model on June 1, 2020, and keep track of all our hyperparameters, libraries, etc.

   We then try to reproduce the same model on July 1, 2020— but the pipeline breaks or the results differ because the underlying data has changed, which could happen if someone added an extra column or an order of magnitude more data after the initial build.

3. Execution Sequence
4. Parallel Operations

Application Model Management

Setting seeds isn't enough!

Before we deploy our machine learning model, we should ensure that we can reproduce and track the model's performance.

Here are examples of model reproducibility-

1. Library Versioning
2. Data Evolution
3. Execution Sequence:
   If a data scientist hands over their code, we should be able to run it top-to bottom without error.

   However, data scientists are notorious for running things out of order, or running the same stateful cell multiple times, making their results very difficult to reproduce. (They might also check in a copy of the code with different hyperparameters than those used to train the final model!)

4. Parallel Operations

Application Model Management

Setting seeds isn't enough!

Before we deploy our machine learning model, we should ensure that we can reproduce and track the model's performance.

Here are examples of model reproducibility-

1. Library Versioning
2. Data Evolution
3. Execution Sequence
4. Parallel Operations:
   To maximize throughput, GPUs will run many operations in parallel.

   However, the order of execution is not always guaranteed, which can lead to nondeterministic outputs. This is a known problem with functions like tf.reduce_sum( ) and when aggregating floating-point numbers (which have limited precision): the order in which you add them may generate slightly different results, which can be exacerbated across many iterations.

Application Model Management

The need to ensure reproducibility

Our inability to reproduce your experiments can often be a blocker in getting business units to adopt your model or put it into production.

It is possible for us to build our own in-house tools for tracking our models, data, dependency versions, etc. but it is also challenging because:

- they may become obsolete,
- Brittle (increased difficulty in fixing older software that may appear reliable, but fails badly when presented with unusual data or altered in a seemingly minor way), and
- take significant development effort to maintain.

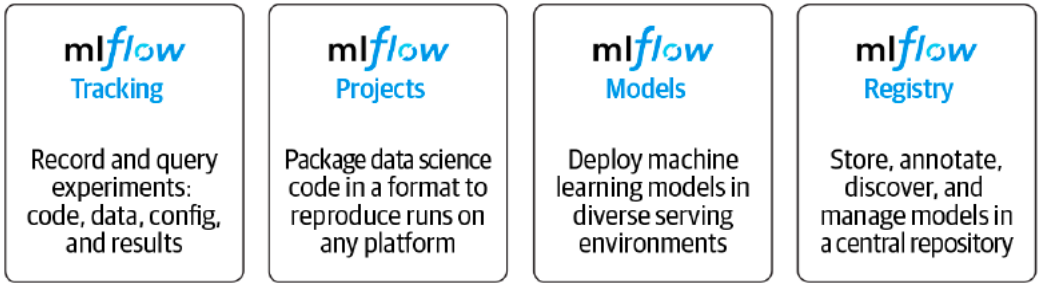
Equally important is having industry-wide standards for managing models so that they can be easily shared with partners (specially if you're consulting across multiple partners).

There are both open source and proprietary tools that can help us with reproducing our machine learning experiments by abstracting away many of these common difficulties.

Model Management

MLflow for reproducibility

MLflow is an open source platform that helps developers reproduce and share experiments, manage models, and much more. It provides interfaces in Python, R, and Java/ Scala, as well as a REST API. Basically, it has four components as described below-



Model Management

MLflow for reproducibility

MLflow is an open source platform that helps developers reproduce and share experiments, manage models, and much more. It provides interfaces in Python, R, and Java/ Scala, as well as a REST API. Basically, it has four components as described below-

- Tracking:
  Provides APIs to record parameters, metrics, code versions, models, and artefacts such as plots, and text;

- Projects:
  A standardized format to package your data science projects and their dependencies to run on other platforms. It helps you manage the model training process;

- Models:
  A standardized format to package models to deploy to diverse execution environments. It provides a consistent API for loading and applying models, regardless of the algorithm or library used to build the model;

- Registry:
  A repository to keep track of model lineage, model versions, stage transitions, and annotations;

Model Management

MLflow for reproducibility

MLflow is an open source platform that helps developers reproduce and share experiments, manage models, and much more. It provides interfaces in Python, R, and Java/ Scala, as well as a REST API.

```
from pyspark.ml import Pipeline
from pyspark.ml.feature import StringIndexer, VectorAssembler
from pyspark.ml.regression import RandomForestRegressor
from pyspark.ml.evaluation import RegressionEvaluator

filePath = """/databricks-datasets/learning-spark-v2/sf-airbnb/
sf-airbnb-clean.parquet"""
airbnbDF = spark.read.parquet(filePath)
(trainDF, testDF) = airbnbDF.randomSplit([.8, .2], seed=42)

categoricalCols = [field for (field, dataType) in trainDF.dtypes
                   if dataType == "string"]
indexOutputCols = [x + "Index" for x in categoricalCols]
stringIndexer = StringIndexer(inputCols=categoricalCols,
                              outputCols=indexOutputCols,
                              handleInvalid="skip")

numericCols = [field for (field, dataType) in trainDF.dtypes
               if ((dataType == "double") & (field != "price"))]
assemblerInputs = indexOutputCols + numericCols
vecAssembler = VectorAssembler(inputCols=assemblerInputs,
                               outputCol="features")

rf = RandomForestRegressor(labelCol="price", maxBins=40, maxDepth=5,
                           numTrees=100, seed=42)

pipeline = Pipeline(stages=[stringIndexer, vecAssembler, rf])
```

Model Management

MLflow for reproducibility

MLflow is an open source platform that helps developers reproduce and share experiments, manage models, and much more. It provides interfaces in Python, R, and Java/ Scala, as well as a REST API.

```
import mlflow
import mlflow.spark
import pandas as pd

with mlflow.start_run(run_name="random-forest") as run:
  # Log params: num_trees and max_depth
  mlflow.log_param("num_trees", rf.getNumTrees())
  mlflow.log_param("max_depth", rf.getMaxDepth())

  # Log model
  pipelineModel = pipeline.fit(trainDF)
  mlflow.spark.log_model(pipelineModel, "model")

  # Log metrics: RMSE and R2
  predDF = pipelineModel.transform(testDF)
  regressionEvaluator = RegressionEvaluator(predictionCol="prediction",
                                            labelCol="price")
  rmse = regressionEvaluator.setMetricName("rmse").evaluate(predDF)
  r2 = regressionEvaluator.setMetricName("r2").evaluate(predDF)
  mlflow.log_metrics({"rmse": rmse, "r2": r2})

  # Log artifact: feature importance scores
  rfModel = pipelineModel.stages[-1]
  pandasDF = (pd.DataFrame(list(zip(vecAssembler.getInputCols(),
                                    rfModel.featureImportances)),
                           columns=["feature", "importance"])
              .sort_values(by="importance", ascending=False))

  # First write to local filesystem, then tell MLflow where to find that file
  pandasDF.to_csv("feature-importance.csv", index=False)
  mlflow.log_artifact("feature-importance.csv")
```

Distributed Parameter Tuning

Introduction to Hyperopt

Even if you do not intend to do distributed inference or do not need MLlib's distributed training capabilities, you can still leverage Spark for distributed hyperparameter tuning.

Hyperopt: Distributed Hyperparameter Optimization

build passing | pypi package 0.2.5 | Anaconda.org 0.2.5

Hyperopt is a Python library for serial and parallel optimization over awkward search spaces, which may include real-valued, discrete, and conditional dimensions.

Distributed Parameter Tuning

Introduction to Hyperopt

Even if you do not intend to do distributed inference or do not need MLlib's distributed training capabilities, you can still leverage Spark for distributed hyperparameter tuning.

There are basically two main ways using which we can scale a spark application using Hyperopt-

1. By running single-machine Hyperopt with a distributed training algorithm (for example, the ones provided by MLlib);
2. By running distributed Hyperopt with single-machine training algorithms with the SparkTrials class;

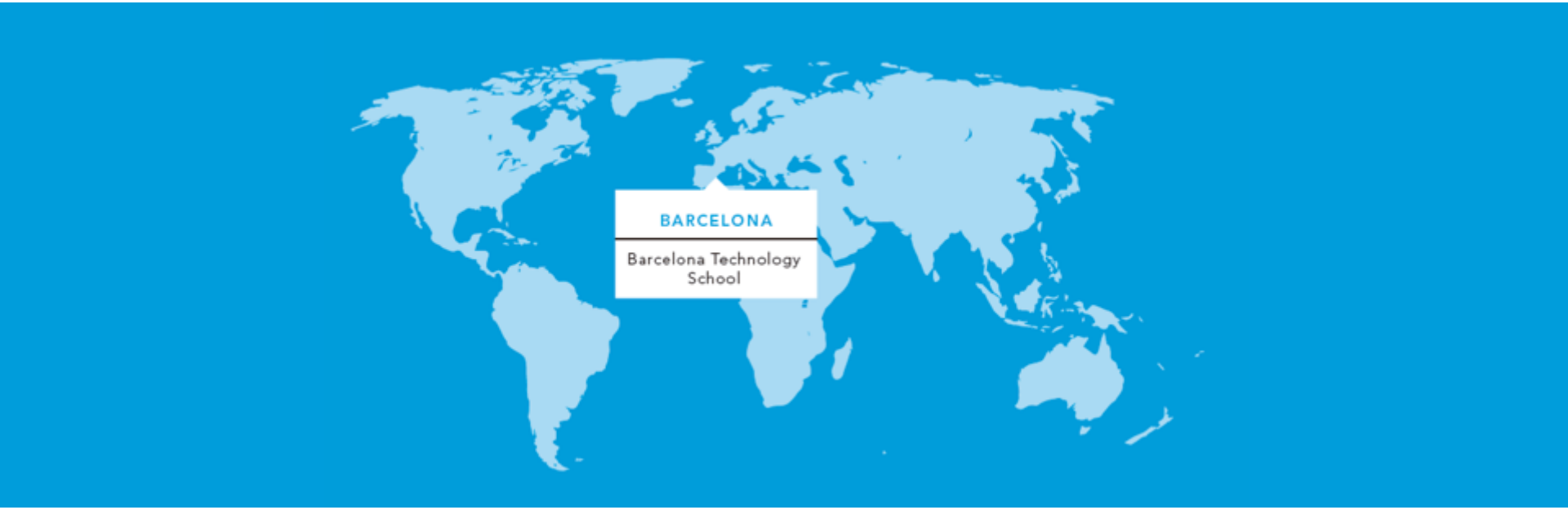
```
import hyperopt

best_hyperparameters = hyperopt.fmin(
  fn = training_function,
  space = search_space,
  algo = hyperopt.tpe.suggest,
  max_evals = 64,
  trials = hyperopt.SparkTrials(parallelism=4))
```

Thank You!

We are grateful for your time and attention..

Questions???


BARCELONA
Barcelona Technology School