

Projet 5:

Développement d'une architecture Big Data complète

Par: RASAMBATRA Freonel Carolio



1. Introduction
2. Présentation du sujet
 - a. Sujet
 - b. Données d'entrées
 - c. Enjeux
3. Architecture de la solution proposée:
 - a. Architecture fonctionnelle
 - b. Architecture technique
4. Mise en oeuvre de la solution proposée:
 - a. System file de message Kafka
 - b. Speed layer: pipeline Storm
 - c. Batch layer: Description de notre stockage et traitement distribué avec Spark
 - d. Serving layer: Speed-view / Batch view
5. Résultats
6. Différent scénarios:

Gestion des pannes- Gestion des erreurs - sécurisation des données - Mise en place d'une architecture robuste
7. Conclusion

1. Introduction:

- Dans les sociétés dont l'activité est pilotée par des informations, les données constituent une ressource d'informations crucial
- Traiter les données en temps-réel permet d'avoir un premier aperçu de la situation ce qui permet d'anticiper une situation avant d'avoir un résultat dont le traitement peut prendre du temps
- L'exploitation optimum des données prend donc en compte:
 - Temps de traitement acceptable ~ temps réel
 - Déploiement: Mobiliser les ressources et les infrastructures capable de supporter des quantités qui peuvent êtres énormes
 - Prendre les précautions nécessaire pour **ne pas perdres de données**
 - Traitement en continue ==> se protéger contre les **pannes et erreurs**

2. Présentation du sujet:

a) Sujet:

Développer une solution complète qui :

- avec l'analyse des hashtags, permettra de créer un tableau de bord affichant le top 10 des mots-clés les plus tendances sur Twitter:
 - depuis une heure
 - à un instant données, n'importe lequel comme lundi dernier entre 7h à 8h
- Origine de nos données API Twitter



b) Présentation des données d'entrées

```
{
  "created_at" : "Thu May 10 17:41:57 +0000 2018" ,
  "id_str" : "994633657141813248" ,
  "text" : "Just another Extended Tweet with more than 140 characters, generated as a documentation example, showing that"
  "display_text_range" : [ 0 , 140 ] ,
  "truncated" : true ,
  "user" : {
    "id_str" : "944480690" ,
    "screen_name" : "FloodSocial"
  } ,
  "extended_tweet" : {
    "full_text" : "Just another Extended Tweet with more than 140 characters, generated as a documentation example, showi"
    "display_text_range" : [ 0 , 249 ] ,
    "entities" : {
      "hashtags" : [ {
        "text" : "documentation" ,
        "indices" : [ 211 , 225 ]
      } , {
        "text" : "parsingJSON" ,
        "indices" : [ 226 , 238 ]
      } , {
        "text" : "GeoTagged" ,
        "indices" : [ 239 , 249 ]
      } ]
    }
  } ,
  "entities" : {
    "hashtags" : [ ]
  }
}
```

c) Les enjeux: architecture robuste

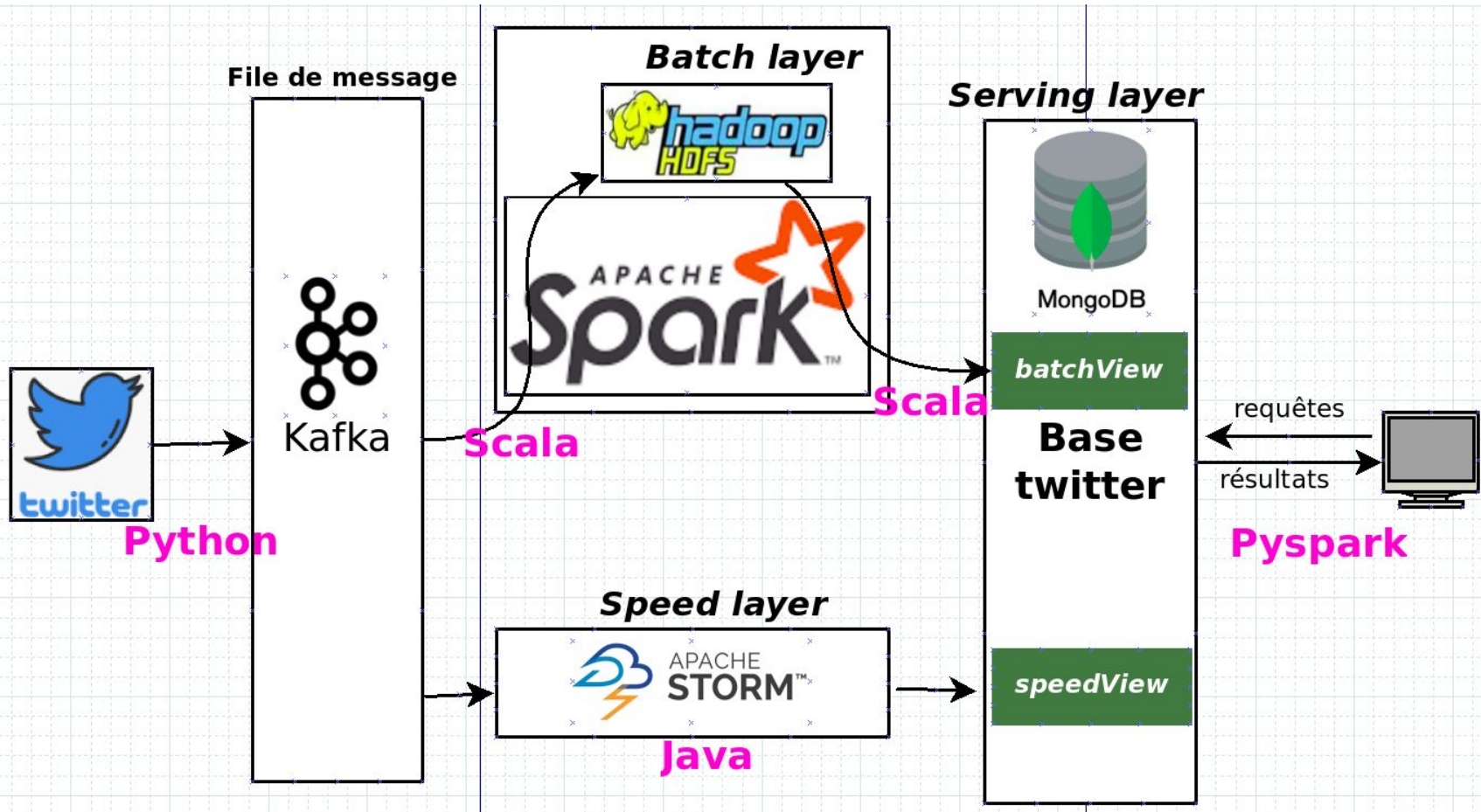
1. Si besoins, la solution permettra de recalculer les tendances à un instant donné:
 - Solution de stockage : prendre toutes les précautions nécessaires pour sécuriser nos données: ex: contre fausse manipulation ou piratage
 - Calcul de gros volume de données: Spark
2. Pas de perte de données:
 - Système de file de message: **Kafka**
 - data lake: batch layer
3. Visualiser les dernières tendances (données fraîches): Speed-view avec speed layer
4. Architecture robuste: No single point of failure:
 - Solution distribuée pour ne pas dépendre d'une seule machine (prévoir panne) et aussi permettant de passer à l'échelle :
 - stockage: HDFS, MongoDB
 - Traitement en temps-réel: Storm
 - Calcul: Spark
 - File de message: Kafka



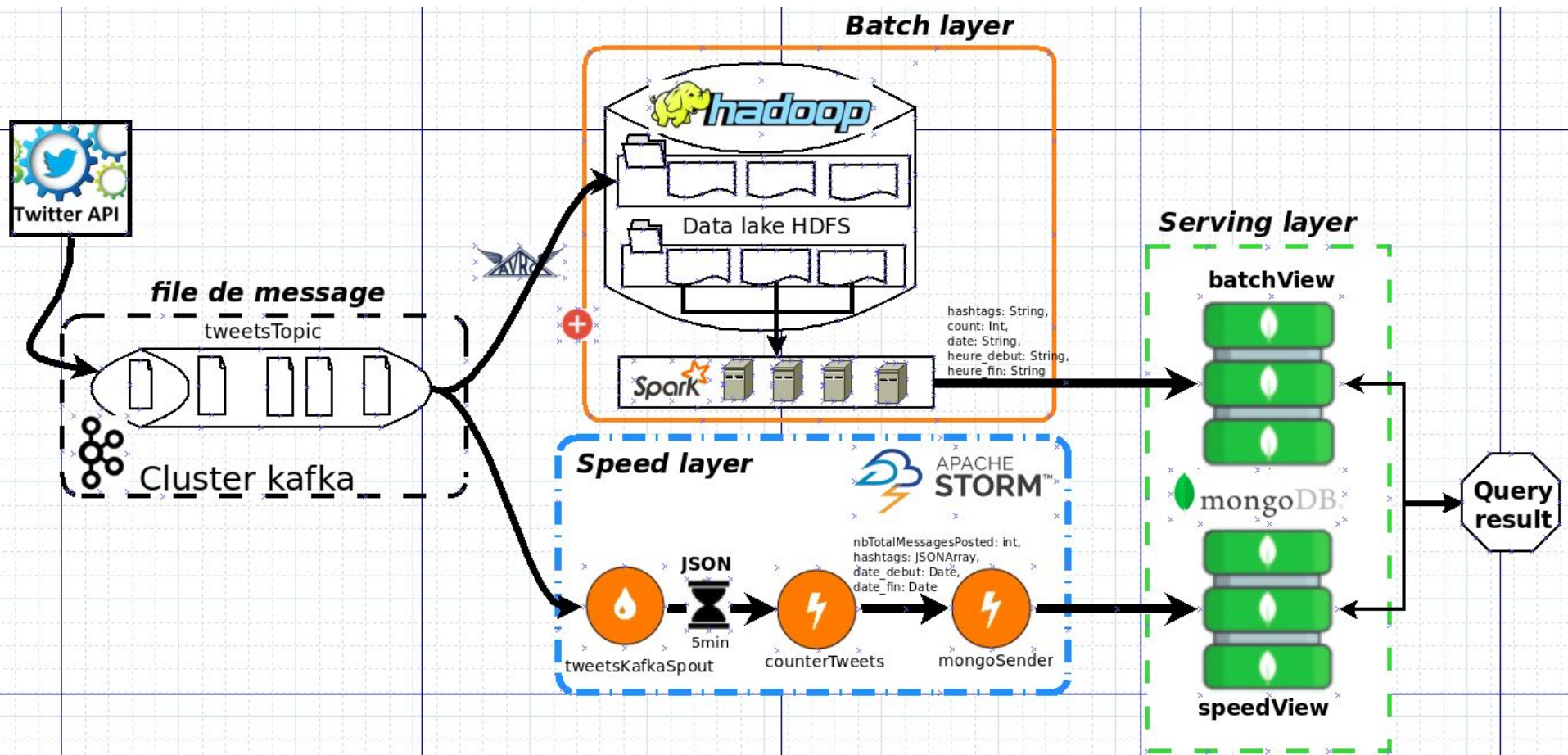
3. Description de l'architecture



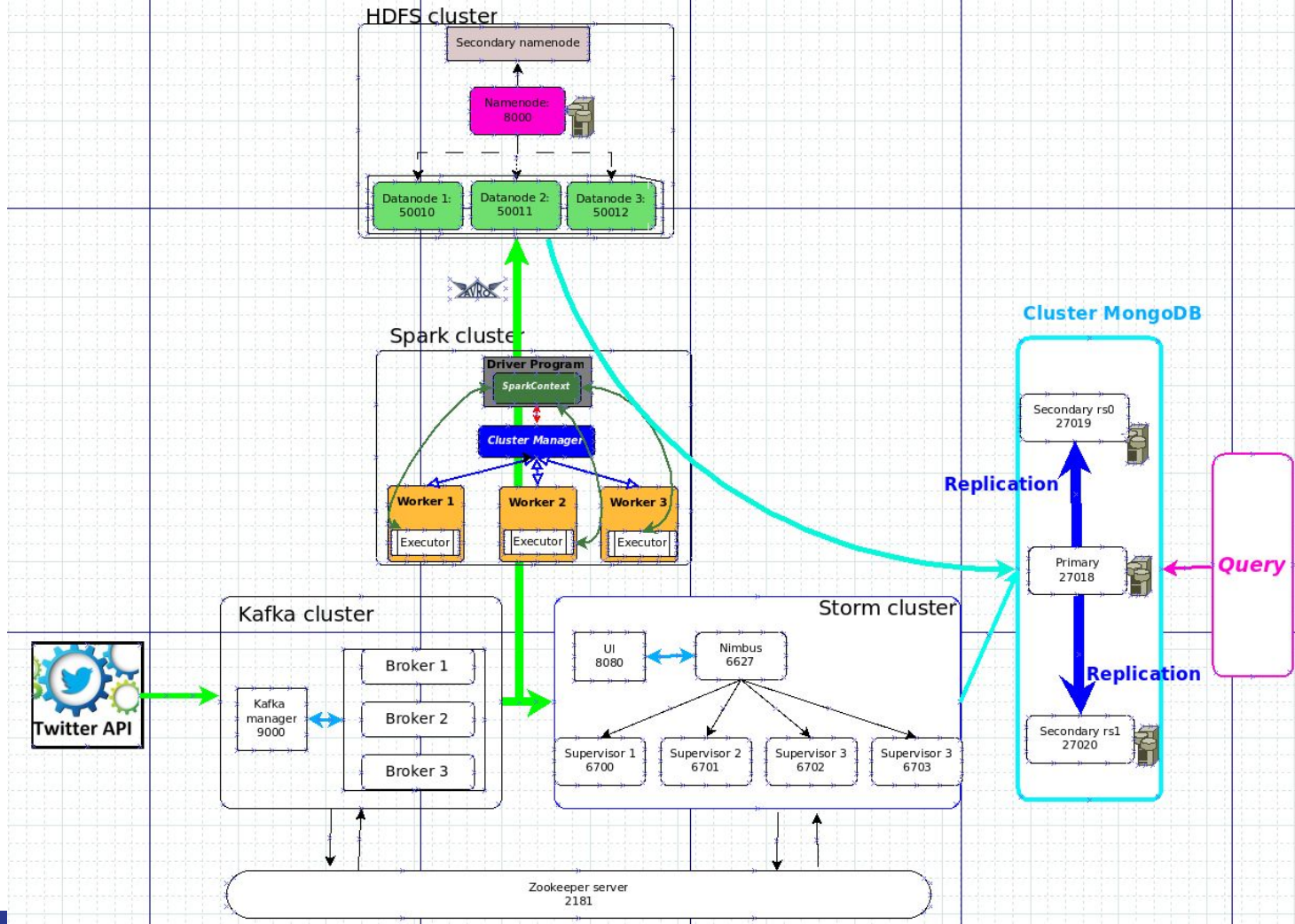
a) Flux de donnée: Architecture **Lambda**:



a) Architecture fonctionnelle:



b) Architecture technique:

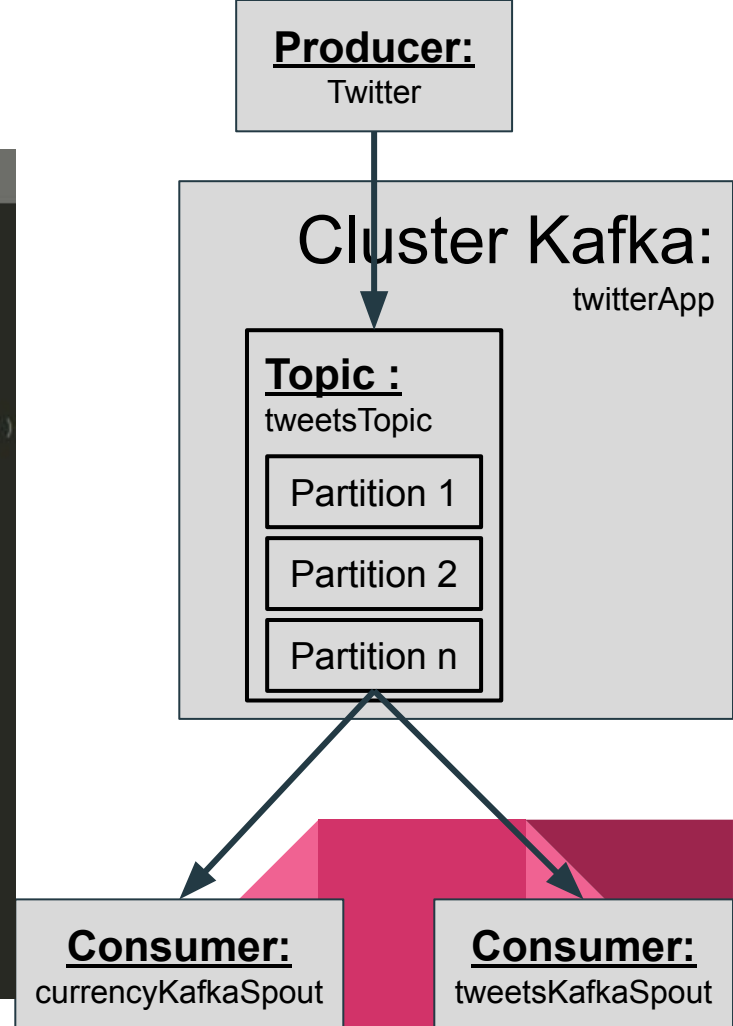


4. Mise en oeuvre de la solution proposée



a) Système file de message: Kafka

```
9
10 producer = KafkaProducer(bootstrap_servers="localhost:9092")
11 # Put here your Twitter API credentials obtained at https://apps.twitter.com/
12
13 # Note: you need a Twitter account to create an app.
14 with open("config.json") as fichierDeConfig:
15     config = json.load(fichierDeConfig)
16
17 #oauth = twitter.OAuth("token", "token_secret", "consumer_key", "consumer_secret")
18 oauth = twitter.OAuth(config["accessToken"],\
19                       config["accessTokenSecret"],\
20                       config["apiKey"],\
21                       config["apiSecretKey"])
22
23 t = twitter.TwitterStream(auth=oauth)
24
25
26 sample_tweets_in_english = t.statuses.sample(language="en")
27
28 for tweet in sample_tweets_in_english:
29     if "delete" in tweet:
30         # Deleted tweet events do not have any associated text
31         continue
32
33     producer.send("tweetsTopic", json.dumps(tweet).encode())
34
35
36
37
38
```



a) Système file de message: Kafka

Topics

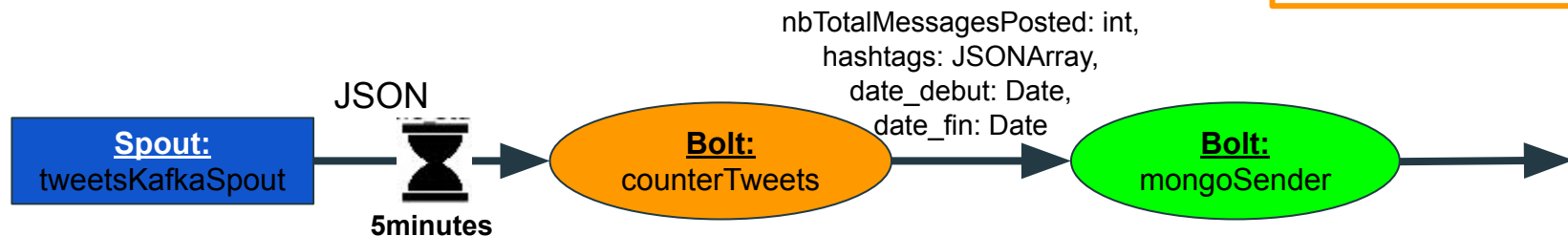
Show 10 entries

Search:

Topic	# Partitions	# Brokers	Brokers Spread %	Brokers Skew %	Brokers Leader Skew %	# Replicas	Under Replicated %	Producer Message/Sec	Summed Recent Offsets
__consumer_offsets	50	1	100	0	0	1	0	0.00	42
tweetsTopic	1	1	100	0	0	1	0	12.68	137 411

b) speed-layer: pipeline Storm (java)

Topology storm:



StatTweetBolt:

1. Parse
2. LinkedHashMap <Str, int>
3. JSONArray[{htag: str, qtité: int}]

```
private static final String url = "mongodb://localhost:27017/twitter";
private static final String collectionName = "speedView";

KafkaSpoutConfig.Builder<String, String> spoutConfigBuilderTwit = KafkaSpoutConfig
    .builder("localhost:9092", "tweetsTopic");
KafkaSpoutConfig<String, String> spoutConfigTwit = spoutConfigBuilderTwit.build();
builder.setSpout("tweetsKafkaSpout", new KafkaSpout<String, String>(spoutConfigTwit));

// Traitement des Tweets
builder.setBolt("counterTweets", new StatTweetBolt().withTumblingWindow(BaseWindowedBolt.Duration.minutes(5)))
    .shuffleGrouping("tweetsKafkaSpout");
// Envoi au serveur mongoDB
builder.setBolt (
    "mongoSender",
    new MongoUpdateBolt(
        url,
        collectionName,
        new SimpleQueryFilterCreator().withField("date_debut"),
        new SimpleMongoUpdateMapper().withFields("date_debut", "date_fin", "hashtags", "nbTotalMessagesPosted")
    )
    .withUpsert(true)
)
.shuffleGrouping("counterTweets");
```

b) speed-layer: pipeline Storm

Storm UI:

Topology actions

Activate

Deactivate

Rebalance

Kill

Debug

Stop Debug

Change Log Level

Topology stats

Window	Emitted	Transferred	Complete latency (ms)	Acked	Failed
3h 0m 0s	170540	170400	155729,377	28720	
10m 0s	48705	48685	149995,449	8060	
1d 0h 0m 0s	170540	170400	155729,377	28720	
All time	170540	170400	155729,377	28720	

Spouts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Complete latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
tweetsKafkaSpout	1	1	59620	59600	155729.37674094707	28720	0				

Showing 1 to 1 of 1 entries

Bolts (All time)

Search:

Id	Executors	Tasks	Emitted	Transferred	Capacity (last 10m)	Execute latency (ms)	Executed	Process latency (ms)	Acked	Failed	Error Host	Error Port	Last error	Error Time
_acker	1	1	28760	28740	0,000	0,006	111880	0,003	111880	0				
counterTweets	1	1	53400	53340	0,000	0,001	29800	294231,573	53360	0				
mongoSender	1	1	28760	28720	0,126	3787,000	20	824,000	20	0				

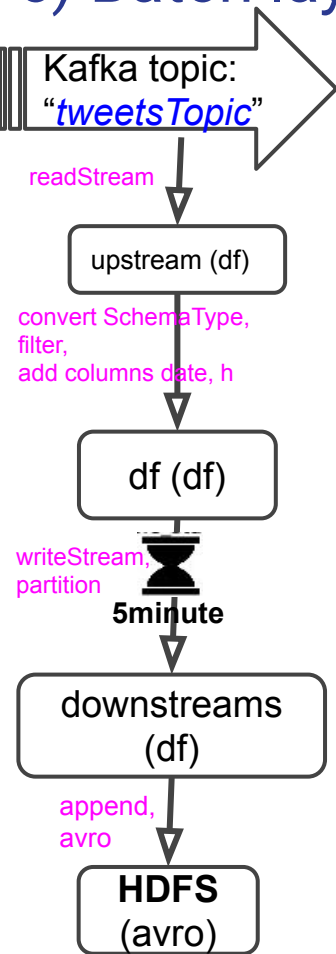
Showing 1 to 3 of 3 entries

Worker Resources

Search: Toggle Components

Host	Supervisor Id	Port	Uptime	Num executors	Assigned Mem (MB)	Components
maiky-ThinkPad-Edge-E330	fd7b2071-37d-4d11-943c-3da990b4c777-127.0.1.1	6701	36m 15s	4	512	4 components

c) Batch layer: Connection kafka->HDFS : Spark Streaming (Scala)



```
1 // Je définis l'adresse de HDFS
2 val addr_hdfs = "hdfs://localhost:8000"
3 val checkpointLocation: String = addr_hdfs + "/tmp/checkpoint-" + UUID.randomUUID.toString
4
5 // On parse le schéma avro pour le convertir en Structtype
6 //Le dataframe obtenu sera calqué à l'image de notre schéma avro
7 val avroSchema = new String(Files.readAllBytes(Paths.get("./schemaTweets.avsc")))
8 val parser: Schema.Parser = new Schema.Parser()
9 val avroSchemaParsed: Schema = parser.parse(avroSchema)
10 val schemaType = SchemaConverters.toSqlType(avroSchemaParsed)
11   .dataType.asInstanceOf[StructType]
12
13 // Lecture du flux kafka
14 val upstream = spark.readStream.format("kafka")
15   .option("kafka.bootstrap.servers", "localhost:9092")
16   .option("subscribe", "tweetsTopic")
17   .load().selectExpr("CAST(value AS STRING)")
18
19 // Conversion de valeur reçu de kafka en fonction du schéma avro,
20 // filter les messages sans hashtags
21 // Ajout de colonne jour et heure pour ranger les données dans HDFS
22 val df = upstream
23   .select(from_json(col("value"), schemaType).as("json")).select("json.*")
24   .filter(size($"json.entities.hashtags")>0)
25   .withColumn("timestamp", col("timestamp_ms").cast(DataTypes.LongType).$div(1000).cast(DataTypes.TimestampType))
26   .withColumn("date", col("timestamp").cast(DataTypes.DateType))
27   .withColumn("heure", hour(col("timestamp")))
28   .drop("timestamp")
29
30 // Ecriture des flux dans HDFS après un temps
31 val downstream = df.writeStream
32   .partitionBy("date", "heure")
33   .format("avro")
34   .option("path", addr_hdfs + "/data/twitter/full")
35   .outputMode("append")
36   .trigger(ProcessingTime(300.seconds))
37   .option("checkpointLocation", checkpointLocation)
38   .start()
39
40 downstream.awaitTermination()
```


c) Batch layer: Description du stockage et sérialisation:

Schéma avro: "schemaTweets.avsc"

```
1 {
2   "namespace": "twitter.mavob",
3   "type": "record",
4   "name": "Tweets",
5   "fields": [
6     {"name": "created_at", "type": "string"},
7     {"name": "id", "type": "long"},
8     {"name": "text", "type": "string"},
9
10    {
11      "name": "entities",
12      "type": {
13        "type": "record",
14        "name": "entite",
15        "fields": [
16          {
17            "name": "hashtags",
18            "type": {
19              "type": "array",
20              "items": [
21                {
22                  "type": "record",
23                  "name": "item",
24                  "fields": [
25                    {"name": "text", "type": ["string", "null"]},
26                    {"name": "indices", "type": {"type": "array", "items": ["int", "null"]}}
27                  ],
28                  "null": "null"
29                }
30              ]
31            }
32          }
33        ]
34      }
35    },
36    {"name": "lang", "type": ["string", "null"]},
37    {"name": "timestamp_ms", "type": ["string", "null"]}
38  ]
39 }
40 }
41 }
```

```
1 /data/
2   .snapshot/*
3   twitter/
4
5       schemaTweets.avsc
6   full/
7       date=2020-09-03/
8       date=2020-09-04/
9       date=2020-09-05/
10
11           heure=0/
12           *.avro
13
14           heure=1/
15           *.avro
16
17           heure=23/
18           *.avro
19
20       test/
21           *.avro
22 /tmp/
23 *
```

Structure HDFS

c) Batch layer: Description du stockage et sérialisation:

Browse Directory

/data/twitter/full/date=2020-09-02/heure=22

Go!



Show 25 entries

Search:

<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name	
<input type="checkbox"/>	-rw-r--r--	maiky	supergroup	1.85 KB	Sep 02 22:16	3	128 MB	part-00000-059e10ba-ea73-44c3-afed-826cdf87e26d.c000.avro	
<input type="checkbox"/>	-rw-r--r--	maiky	supergroup	57.72 KB	Sep 02 22:20	3	128 MB	part-00000-953ffa27-7634-49b7-ba78-bfad53d2854a.c000.avro	
<input type="checkbox"/>	-rw-r--r--	maiky	supergroup	63.73 KB	Sep 02 22:25	3	128 MB	part-00000-bf5ddb96-248c-4f49-915d-96bfb2025b3a.c000.avro	

c) Batch layer: Description du stockage et sérialisation:

```
maiky@maiky-ThinkPad-Edge-E330:~$ hdfs dfs -ls /data/twitter/full/date=2020-09-05
```

Found 18 items

drwxrwxrwx	- maiky supergroup	0	2020-09-05	01:05	/data/twitter/full/date=2020-09-05/heure=0
drwxrwxrwx	- maiky supergroup	0	2020-09-05	02:05	/data/twitter/full/date=2020-09-05/heure=1
drwxrwxrwx	- maiky supergroup	0	2020-09-05	11:05	/data/twitter/full/date=2020-09-05/heure=10
drwxrwxrwx	- maiky supergroup	0	2020-09-05	12:05	/data/twitter/full/date=2020-09-05/heure=11
drwxrwxrwx	- maiky supergroup	0	2020-09-05	13:05	/data/twitter/full/date=2020-09-05/heure=12
drwxrwxrwx	- maiky supergroup	0	2020-09-05	14:05	/data/twitter/full/date=2020-09-05/heure=13
drwxrwxrwx	- maiky supergroup	0	2020-09-05	15:05	/data/twitter/full/date=2020-09-05/heure=14
drwxrwxrwx	- maiky supergroup	0	2020-09-05	15:45	/data/twitter/full/date=2020-09-05/heure=15
drwxrwxrwx	- maiky supergroup	0	2020-09-05	17:05	/data/twitter/full/date=2020-09-05/heure=16
drwxrwxrwx	- maiky supergroup	0	2020-09-05	17:55	/data/twitter/full/date=2020-09-05/heure=17
drwxrwxrwx	- maiky supergroup	0	2020-09-05	03:05	/data/twitter/full/date=2020-09-05/heure=2
drwxrwxrwx	- maiky supergroup	0	2020-09-05	04:05	/data/twitter/full/date=2020-09-05/heure=3
drwxrwxrwx	- maiky supergroup	0	2020-09-05	05:05	/data/twitter/full/date=2020-09-05/heure=4
drwxrwxrwx	- maiky supergroup	0	2020-09-05	06:05	/data/twitter/full/date=2020-09-05/heure=5
drwxrwxrwx	- maiky supergroup	0	2020-09-05	07:05	/data/twitter/full/date=2020-09-05/heure=6
drwxrwxrwx	- maiky supergroup	0	2020-09-05	08:05	/data/twitter/full/date=2020-09-05/heure=7
drwxrwxrwx	- maiky supergroup	0	2020-09-05	08:47	/data/twitter/full/date=2020-09-05/heure=8
drwxrwxrwx	- maiky supergroup	0	2020-09-05	10:05	/data/twitter/full/date=2020-09-05/heure=9

```
maiky@maiky-ThinkPad-Edge-E330:~$ hdfs dfs -cat /data1/twitter/full/date=2020-09-02/heure=22/part-000000-059e10ba-ea73-44c3-afed-826cdf87e26d.c000.avro
```

```
ObjectNodevro.schema{"type":"record","name":"topLevelRecord","fields":[{"name":"created_at","type":["string","null"]},{name":"id","type":["long","null"]},{name":"text","type":["string","null"]},{name":"entities","type":[{"type":"record","name":"entities","namespace":"topLevelRecord","fields":[{"name":"hashtags","type":[{"type":"array","items":[{"type":"record","name":"hashtags","namespace":"topLevelRecord.entities","fields":[{"name":"text","type":["string","null"]},{name":"indices","type":[{"type":"array","items":["int","null"],"null"}],"null"}],"null"}],"null"}],"null"}],"name":"lang","type":["string","null"]},{name":"timestamp_ms","type":["string","null"]}]}org.apache.spark.version3.0.0vro.codec
```

shappyy@Q:~\$ cat /dev/urandom | tr -dc 'a-z0-9' | fold -w 64 | xargs -n 1 sh -c 'curl -s -X POST -H "Content-Type: application/json" -d {"text": "i<Wed Sep 02 20:15:17 +0000 2020\$\$\$@JENFL23: Medicare For All is government funded. NOT gover run, healthcare.

U,ForAll[0][0][0][0][0][0]en[0]599077717659♦♦[0][0] ♦♦[0][0][0][0]Ielliedits: serkan: where should we go sirius?

sirius: to see mom ofc

se = i think so too

6PB • #özgüvenS...J...Hl, @Z903! Can you please play #IceCream by Blackpink x Selena Gomez? Thank.%
8658

•Episode 10•

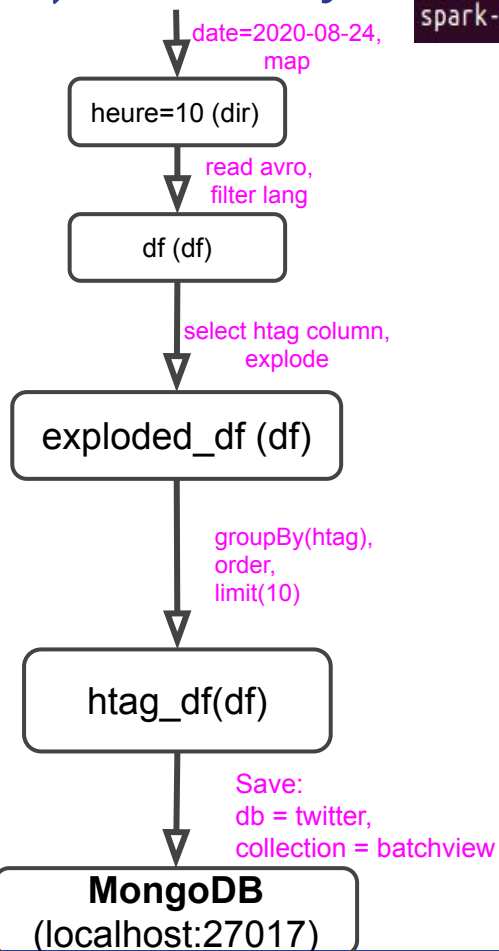
#B 64s #CanYaman #ÖzgeGürel #Ez

OrAtasoyOriginal <https://t.co/b0vyLSznPN>

[illegible]

c) Batch layer: Tweet processing HDFS->mongoDB : Spark (Scala)

```
spark-submit --class App --master local ./target/scala-2.12/tweet_batchprocessing_2.12-0.1.jar date=2020-09-05
```



```
val date_jour = "/data/twitter/full/" + args(0) //"/data/twitter/fu
val fs = org.apache.hadoop.fs.FileSystem
    .get(new URI(port_hdfs), spark.sparkContext.hadoopConfiguration)
fs.listStatus(new Path(date_jour))
    .filter(_.isDirectory)
    .map(_.getPath)
    .foreach(unDir => traiterUneheureDonnee(unDir.toString))
```

```
// Lecture des données à une heure donnée dans hdfs
val df = spark.read
    .format("avro") // format avro
    .load(hourPathInHdfs)
    .filter("lang == 'en'")

// Mettre chaque élément de la liste en Rows
val exploded_df = df.selectExpr("explode(entities.hashtags)")
exploded_df.persist()

// Grouper, compter et ordonner les hashtags selon leur nombre
val htag_df = exploded_df.select("col.text")
    .withColumnRenamed("text", "hashtags")
    .groupBy("hashtags").count()
    .orderBy(col("count").desc)
    .limit(10)
    .withColumn("date", lit(date).cast(StringType))
    .withColumn("heureDebut", lit(heureDebut.toString + "h:00m:00s"))
    .withColumn("heureFin", lit(heureFin.toString + "h:00m:00s"))

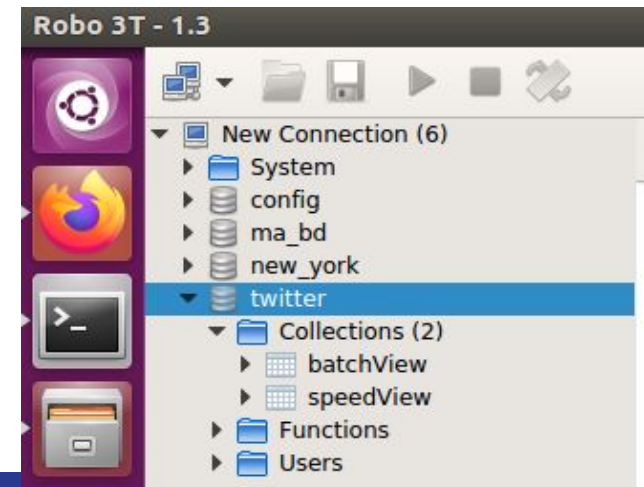
// Ecriture des top dix htags pendant une heure dans mongodb
htag_df.write.format("mongo") //com.mongodb.spark.sql.DefaultSource
    .mode("append")
    .option("uri", "mongodb://localhost:27017")
    .option("database", "twitter")
    .option("collection", "batchView")
    .save()
```

d) Serving layer: MongoDB :

```
removerOld_SpeedView.js x tweetsProducer.py x App.java x
1 use twitter
2 show collections
3 db.getCollection('speedView').deleteMany({date_debut:
4 {
5     $lt: new Date(new Date().setDate(new Date().getDate()-1))
6 }
7 })
```

speedView 0.052 sec.

Key	Value	Type
(1) ObjectId("5f53444c5e061dccdecc68a4")	{ 5 fields }	Object
_id	ObjectId("5f53444c5e061dccdecc68a4")	ObjectId
date_debut	2020-09-05 07:44:40.657Z	Date
date_fin	2020-09-05 07:54:43.666Z	Date
hashtags	[10 elements]	Array
[0]	{ 2 fields }	Object
hashtagUsed	RRBExamDates	String
quantite	1066	Int32
[1]	{ 2 fields }	Object
hashtagUsed	speakup	String
quantite	379	Int32
[2]	{ 2 fields }	Object
[3]	{ 2 fields }	Object
[4]	{ 2 fields }	Object
[5]	{ 2 fields }	Object
[6]	{ 2 fields }	Object
[7]	{ 2 fields }	Object
[8]	{ 2 fields }	Object
[9]	{ 2 fields }	Object
nbTotalMessagesPosted	8524	Int32



d) Serving layer: MongoDB : batchView

batchView 1.31 sec.		
Key	Value	Type
▼ (1) ObjectId("5f51c406db4d1a2a3c793cd7")	{ 6 fields }	Object
_id	ObjectId("5f51c406db4d1a2a3c793cd7")	ObjectId
hashtags	JEEFailedPostponeNEET	String
count	87	Int64
date	2020-09-03	String
heureDebut	16h:00m:00s	String
heureFin	17h:00m:00s	String

```
df2 = spark.sql("""
SELECT hashtags, count FROM premier_df df1
WHERE df1.date == '2020-09-03'
AND df1.heureDebut == '19h:00m:00s'
AND df1.heureFin == '20h:00m:00s'
""")
```

_id	count	date	hashtags	heureDebut	heureFin
[5f51c406db4d1a2a...]	87	2020-09-03	JEEFailedPostpone...	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	76	2020-09-03	JK	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	56	2020-09-03	BTS	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	54	2020-09-03	WeLoveRCB	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	43	2020-09-03	PMModi RozgarDo	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	32	2020-09-03	방탄소년단	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	23	2020-09-03	speakup	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	23	2020-09-03	ForeverWithMarkTu...	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	21	2020-09-03	MalixBrightWin	16h:00m:00s	17h:00m:00s
[5f51c406db4d1a2a...]	21	2020-09-03	NintendoSwitch	16h:00m:00s	17h:00m:00s
[5f51c40adb4d1a2a...]	550	2020-09-03	ForeverWithMarkTu...	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	205	2020-09-03	아가새의소중한마크 생일축하해	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	101	2020-09-03	JEEFailedPostpone...	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	99	2020-09-03	샤이니	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	99	2020-09-03	태민	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	99	2020-09-03	TAEMIN	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	98	2020-09-03	GOT7	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	90	2020-09-03	JK	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	81	2020-09-03	MarkTuan	17h:00m:00s	18h:00m:00s
[5f51c40adb4d1a2a...]	56	2020-09-03	WeLoveRCB	17h:00m:00s	18h:00m:00s

d) Serving layer: MongoDB : batchView

batchView 1.31 sec.							0	50
	_id	hashtags	count	date	heureDebut	heureFin		
1	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cd7")	<input type="checkbox"/> JEEFailedPostponeNEET	<input type="checkbox"/> 87	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
2	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cd8")	<input type="checkbox"/> JK	<input type="checkbox"/> 76	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
3	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cd9")	<input type="checkbox"/> BTS	<input type="checkbox"/> 56	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
4	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cda")	<input type="checkbox"/> WeLoveRCB	<input type="checkbox"/> 54	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
5	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cdb")	<input type="checkbox"/> PMModi_RozgarDo	<input type="checkbox"/> 43	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
6	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cdc")	<input type="checkbox"/> 방탄소년단	<input type="checkbox"/> 32	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
7	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cdd")	<input type="checkbox"/> speakup	<input type="checkbox"/> 23	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
8	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cde")	<input type="checkbox"/> ForeverWithMarkTuanD...	<input type="checkbox"/> 23	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
9	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793cdf")	<input type="checkbox"/> MalixBrightWin	<input type="checkbox"/> 21	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
10	<input type="checkbox"/> ObjectId("5f51c406db4d1a2a3c793ce0")	<input type="checkbox"/> NintendoSwitch	<input type="checkbox"/> 21	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 16h:00m:00s	<input type="checkbox"/> 17h:00m:00s		
11	<input type="checkbox"/> ObjectId("5f51c40adb4d1a2a3c793ce1")	<input type="checkbox"/> ForeverWithMarkTuanD...	<input type="checkbox"/> 550	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 17h:00m:00s	<input type="checkbox"/> 18h:00m:00s		
12	<input type="checkbox"/> ObjectId("5f51c40adb4d1a2a3c793ce2")	<input type="checkbox"/> 아가새의소중한마크_생일축...	<input type="checkbox"/> 205	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 17h:00m:00s	<input type="checkbox"/> 18h:00m:00s		
13	<input type="checkbox"/> ObjectId("5f51c40adb4d1a2a3c793ce3")	<input type="checkbox"/> JEEFailedPostponeNEET	<input type="checkbox"/> 101	<input type="checkbox"/> 2020-09-03	<input type="checkbox"/> 17h:00m:00s	<input type="checkbox"/> 18h:00m:00s		

d) Serving layer: MongoDB : speedView


































```
df1 = df.filter("date_debut > timestamp'2020-09-05 10:00:00' AND date_fin < timestamp'2020-09-05 11:00:00')\n        .select("hashtags")
```

speedView 0.052 sec.

Key	Value	Type
▼ (1) ObjectId("5f53444c5e061dccdecc68a4")	{ 5 fields }	Object
_id	ObjectId("5f53444c5e061dccdecc68a4")	ObjectId
date_debut	2020-09-05 07:44:40.657Z	Date
date_fin	2020-09-05 07:54:43.666Z	Date
▼ hashtags	[10 elements]	Array
▼ [0]	{ 2 fields }	Object
hashtagUsed	RRBExamDates	String
# quantite	1066	Int32
▼ [1]	{ 2 fields }	Object
hashtagUsed	speakup	String
# quantite	379	Int32
▶ [2]	{ 2 fields }	Object
▶ [3]	{ 2 fields }	Object
▶ [4]	{ 2 fields }	Object
▶ [5]	{ 2 fields }	Object
▶ [6]	{ 2 fields }	Object
▶ [7]	{ 2 fields }	Object
▶ [8]	{ 2 fields }	Object
▶ [9]	{ 2 fields }	Object
# nbTotalMessagesPosted	8524	Int32

```
root
|-- _id: struct (nullable = true)
|   |-- oid: string (nullable = true)
|-- date_debut: timestamp (nullable = true)
|-- date_fin: timestamp (nullable = true)
|-- hashtags: array (nullable = true)
|   |-- element: struct (containsNull = true)
|   |   |-- hashtagUsed: string (nullable = true)
|   |   |-- quantite: integer (nullable = true)
|-- nbTotalMessagesPosted: integer (nullable = true)
```


d) Serving layer: MongoDB : speedView

speedView 0.052 sec.					
	_id	date_debut	date_fin	hashtags	nbTotalMessage
1	<input type="checkbox"/> ObjectId("5f53444c5e061dccdecc68a4")	 2020-09-05 07:44:40.657Z	 2020-09-05 07:54:43.666Z	 [10 elements]	# 8524
2	<input type="checkbox"/> ObjectId("5f5345765e061dccdecc68d...")	 2020-09-05 07:54:43.657Z	 2020-09-05 07:59:43.666Z	 [10 elements]	# 4378
3	<input type="checkbox"/> ObjectId("5f53479f5e061dccdecc6935")	 2020-09-05 07:59:43.659Z	 2020-09-05 08:08:56.664Z	 [10 elements]	# 6765
4	<input type="checkbox"/> ObjectId("5f5348ca5e061dccdecc6973")	 2020-09-05 08:08:56.666Z	 2020-09-05 08:13:56.666Z	 [10 elements]	# 4066
5	<input type="checkbox"/> ObjectId("5f5349f75e061dccdecc69b4")	 2020-09-05 08:13:56.657Z	 2020-09-05 08:18:56.664Z	 [10 elements]	# 3995
6	<input type="checkbox"/> ObjectId("5f534b225e061dccdecc69e...")	 2020-09-05 08:18:56.664Z	 2020-09-05 08:23:56.666Z	 [10 elements]	# 3938
7	<input type="checkbox"/> ObjectId("5f534c4e5e061dccdecc6a44")	 2020-09-05 08:23:56.660Z	 2020-09-05 08:28:56.665Z	 [10 elements]	# 4037
8	<input type="checkbox"/> ObjectId("5f534d7b5e061dccdecc6a9...")	 2020-09-05 08:28:56.658Z	 2020-09-05 08:33:56.666Z	 [10 elements]	# 4070
9	<input type="checkbox"/> ObjectId("5f534ea65e061dccdecc6ace")	 2020-09-05 08:33:57.657Z	 2020-09-05 08:38:56.666Z	 [10 elements]	# 3958
10	<input type="checkbox"/> ObjectId("5f534fd25e061dccdecc6b0f")	 2020-09-05 08:38:56.657Z	 2020-09-05 08:43:56.666Z	 [10 elements]	# 3956
11	<input type="checkbox"/> ...	 2020-09-05	 2020-09-05	 [10 elements]	#

5. Résultats:

```
bmit ./requetBatchView.py 2020-09-05 10
*****
Le 2020-09-05 entre 10:00:00 et 11:00:00, ci-dessous les hashtags les plus utilisés:
+-----+-----+
|          | hashtags|count|
+-----+-----+
|          | RRBExamDates| 5197|
|          | speakup| 2020|
|          | SpeakUpFor69000Te...| 489|
|          | Dynamite300M| 300|
|          | SpeakUpForSSCRail...| 289|
|          | 5Baje5Minutes| 278|
|          | rrbexamdates| 259|
|          | BTS| 225|
|          | BiharBole_RozgarDo| 211|
|          | rrbexamdate| 187|
+-----+-----+
```

```
bmit ./requetSpeedView.py
*****
Le 2020-09-06 entre 0:00:00 et 1:00:00, ci-dessous les hashtags les plus utilisés:
+-----+-----+
|          | hashtagUsed|sum(quantitee)|
+-----+-----+
|          | ArtistoftheSummer| 782|
|          | BBNaija| 246|
|          | BBNaija| 126|
|          | TurnUpWithLaycon| 88|
|          | BTS| 74|
|          | bbnaija| 69|
|          | SayangAdminShopee| 31|
|          | 23YrsofSENSATIONA...| 29|
|          | BBNaijaShallWe| 27|
|          | Dumbkirk| 25|
+-----+-----+
```



6. Présentation des différentes scénarios



6- scénarios: Kafka

Cluster	Scénario	Exigences	Solutions proposées
Kafka	Panne d'une ou plusieurs machines (serveur kafka)	Pas de single point of failure	répliquer les données de manière redondantes sur différents serveurs
	MAJ nécessitant redémarrage des machines du cluster	Pas d'interruption de service	avoir plusieurs machines et les redémarrer une par une
	Augmentation quantité de données	Passer à l'échelle de données élevées massives	distribuer les tâches sur plusieurs serveurs Kafka
Zookeeper	Erreur-plantage Zookeeper⇒ s'éteint brusquement	Tolérant au fail-fast	Utiliser des outils de supervision (supervisor) pour gérer zookeeper: redémarrage automatique

6- scénarios: Kafka


Pour ajouter un serveur dans un cluster Kafka:

- Serveur Kafka communique par zookeeper --> ajouter une ou plusieurs machine dans Zookeeper
- Définir le “**broker.id**” (int) des nouveaux server à partir de leur fichiers de configuration “[config/server.properties](#)”. Cet identifiant doit être unique pour chaque serveur.
- Lancer le nouveau serveur Kafka

Répliquer les données:

- L'option **--replication-factor** permet d'augmenter le taux de réplication d'un topic, ici :
`$./bin/kafka-topics.sh --create --zookeeper localhost:2181 --replication-factor 2 --partitions 1 --topic bitcoin-price`
- Si taux de réplication est de **N**, l'architecture permettra de supporter la panne de **N-1** serveurs.



Cluster	Scénario	Exigences	Solutions proposées
Storm	échec d'un traitement de tuples	Pas de perte de message	Gestion des erreurs: réémettre les tuples dont le traitement a échoué
	un worker s'éteint	Pas de single point of failure	Supervisor le redémarre et si répétitif, nimbus ré-affecte ses tâche à un autre worker
	Redémarrage de bolts ou de la topologie	Pas de pertes de données	stocker les tuples en cours de traitement dans une base de données à laquelle pourront accéder tous les bolts
	Augmentation quantité de données	Passer à l'échelle du Big data	distribuer les tâches d'un bolt sur plusieurs --> tuple grouping +  nb de executors
Zookeeper ou Storm	Zookeeper ou worker s'éteint	Pas de single point of failure	<ul style="list-style-type: none"> - Supervisor les redemarre - Nimbus arrêté : worker continuent traitement des taches mais impossible de réaffecter les tâches des workers ou de soumettre de nouvelles topologies - mettre second nimbus en backup

6- scénarios: Storm

Pour re-émettre les tuples dont le traitement a échoué :

- Tracer le spout ayant émis le tuple:
 - en ajoutant un **identifiant aux tuples** émis par les spout ==> Ceci se fait à l'appel de la méthode **emit()** dans le spout
- Les bolts indiquent le succès ou l'échec du traitement de chaque tuple à l'aide des méthodes **ack()** et **fail()**.
- La méthode **fail()** du spout ayant émis un tuple sera appelée dès que le tuple ou l'un de ses descendants sera en échec.

```
...  
@Override  
public void fail(Object msgId) {  
    Values tuple = (Values)msgId;  
  
    // En cas d'erreur, on ré-émet le tuple lui-même  
    outputCollector.emit(tuple, msgId);  
}
```

Parallélisations des tâches:

- augmenter le nb d' executors et optimiser le paramètre **parallelism_hint** aux méthodes **setBolt()** et **setSpout()** :

```
// Sauvegarde du current price dans elasticSearch  
builder.setBolt("savingES-currencyPrice", new EsBolt("current_price/devises", conf), 1)  
    .shuffleGrouping("parseDeviseBolt");
```

- Optimiser les tuples grouping: shuffle, fliefs

6- Sécurisation des données HDFS:

1. No single point of failure:
 - a. 3 datanodes sur lesquels les données sont distribuées en blocs
 - b. Namenode secondaire: préserver les données du namenode en faisant des checkpoints régulièrement (toutes les heures)
2. Snapshot: permet de sauvegarder l'état d'un répertoire à un instant t donné
 - i. Rendre le répertoire `"/data"` snapshotable: `hdfs dfsadmin -allowSnapshot /data`
 - ii. Créer des snapshot régulières au rep `"/data"`: `hdfs dfs -createSnapshot /data`
 - iii. Restaurer: `hdfs dfs -cp -f /data/.snapshot/s20200720-163848.488/* /data/`
3. Interdire l'accès en écriture sur les deux répertoire contenant les données brutes et données sérialisés (master dataset):
 - i. bash: `hdfs dfs -chmod -R ugo-w /data/twitter/master/full`

/data/frwiki/raw								Go!		
Show	25	entries	Search:							
<input type="checkbox"/>	Permission	Owner	Group	Size	Last Modified	Replication	Block Size	Name		
<input type="checkbox"/>	-r--r--r--	maiky	supergroup	11.04 GB	Jul 05 14:11	3	128 MB	frwiki-20200201-pagelinks.sql		
<input type="checkbox"/>	-r--r--r--	maiky	supergroup	70.72 GB	Jul 05 13:56	3	128 MB	frwiki-20200201-stub-meta-history.xml		

b) Sécurisation des données HDFS:

1. Snapshot:

Hadoop	Overview	Datanodes	Datanode Volume Failures	Snapshot	Startup Progress	Utilities ▾
--------	----------	-----------	--------------------------	----------	------------------	-------------

Snapshot Summary

Snapshottable directories: 1

Path	Snapshot Number	Snapshot Quota	Modification Time	Permission	Owner	Group
/data	2	65536	Mon Jul 20 16:38:48 +0200 2020	rwxr-xr-x	maiky	supergroup

Snapshotted directories: 2

Snapshot ID	Snapshot Directory	Modification Time
s20200720-163848.488	/data/.snapshot/s20200720-163848.488	Mon Jul 20 16:38:48 +0200 2020
snapshot1	/data/.snapshot/snapshot1	Mon Jul 20 16:28:21 +0200 2020

scénarios: MongoDB

Cluster	Scénario	Exigences	Solutions proposées
MongoDB	Panne d'une ou plusieurs machines du cluster MongoDB	résistance au pannes	Utilisation de l'architecture replicaset puis d'un arbitre
	MAJ nécessitant redémarrage des machines du cluster	Pas d'interruption de service	avoir plusieurs machines et les redémarrer une par une
	Augmentation quantité de données	Passer à l'échelle de données élevées massives	répartir les charges avec l'architecture du sharding
Zookeep er	Erreur-plantage Zookeeper⇒ s' éteint brusquement	Tolérant au fail-fast	Gérer zookeeper avec un outil de supervision (supervisord): redemarrage automatique

8. Conclusion:

1. On a vu que l'architecture lambda protège de presque toutes les problèmes les plus courantes dans le traitement et stockage des données :
 - a. scénarios catastrophes
 - b. pannes de machine
 - c. utilisation et évolution des données dans le temps
2. Optimisation des applications est cruciale!!
 - Éviter d'encombrer la mémoire pour le traitement sinon pas de données traitées: exemple sur le windowed Bolt de Storm
 - Traiter par lots : par tranche 5 min au lieu d'une heure
 - code optimisé peut faire gagner beaucoup de temps: moins de latence
3. Requêtes sur de gros volumes de données très coûteuses en temps:
 - a. Utiliser *[persist\(\)](#)* avec le stockage local si nécessaire pour éviter de répéter les requêtes dans HDFS et MongoDB
 - b. Intéressant d'optimiser le nombre de machines nœuds

Annexe



a) Structure de notre data lake:

- **/data/** : Contient tous nos données
- **/data.snapshot/s20200720-163848.488/*** : contient le snapshot de tous nos données
- **/data/frwiki/** : contient tout ce qui provient de Wikipedia
- **/data/frwiki/raw/** : contient tous nos données brutes
- **/data/frwiki/raw/frwiki-20200201-pagelinks.sql** : fichier brute contenant les liens entre les pages (11.04 GB)
- **/data/frwiki/raw/frwiki-20200201-stub-meta-history.xml** : fichier brute contenant les informations et historique des pages (70.72 GB)
- **/data/frwiki/frwiki-20200201/** : contient tout ce qui provient des données brutes du 1er février 2020
- **/data/frwiki/frwiki-20200201/master/** : contient tous les jeux de données provenant des données brutes wikipédia du 1er février 2020
- **/data/frwiki/frwiki-20200201/master/pagelink.avsc** : schéma avro utilisé pour sérialiser les liens des pages
- **/data/frwiki/frwiki-20200201/master/pageshistory.avsc** : schéma avro utilisé pour sérialiser l'historique et les informations concernant les pages
- **/data/frwiki/frwiki-20200201/master/full/** : contient tous les données sérialisé (complet)
- **/data/frwiki/frwiki-20200201/master/full/frwiki-20200201-pagelinks-*.avro**: pages links sérialisés
- **/data/frwiki/frwiki-20200201/master/full/frwiki-20200201-stub-meta-history-*.avro** : history sérialisé
- **/data/frwiki/frwiki-20200201/master/test/** : contient des échantillons de jeux de données sérialisés ==> pour faire des tests de traitement ou d'observer des de nos données

Add Cluster

Cluster Name

twitterApp

Cluster Zookeeper Hosts

localhost:2181

Kafka Version

2.1.1

☐ Enable JMX Polling (Set JMX_PORT env variable before starting kafka server)

JMX Auth Username

JMX Auth Password

☐ JMX with SSL

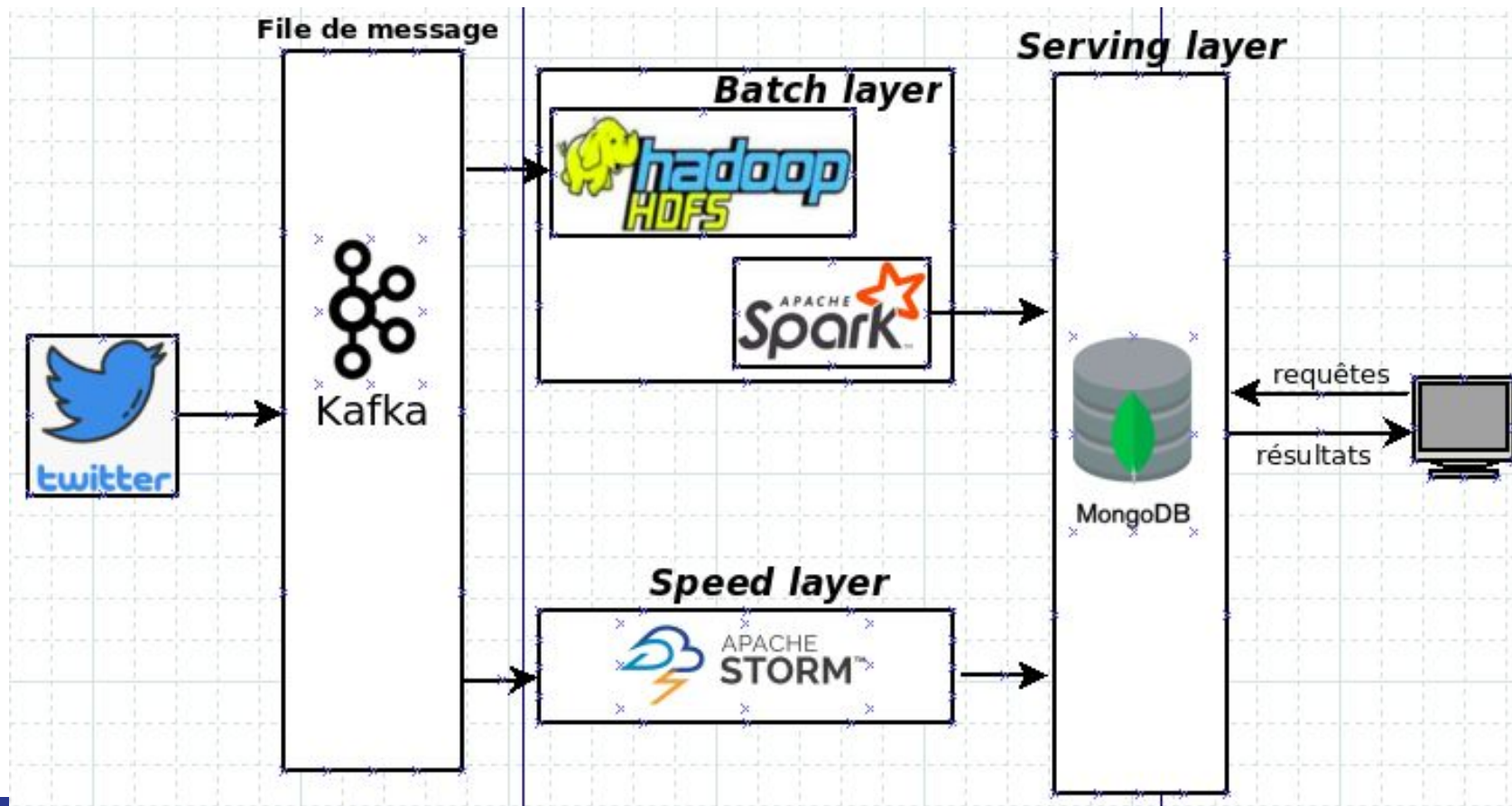
☐ Enable Logkafka

☒ Poll consumer information (Not recommended for large # of consumers if ZK is used for offsets tracking on older Kafka versions)

c) Démarrage kafka-manager:



a) Flux de donnée: Architecture **Lambda**:



b) Architecture technique:

