

Projet Bigdata

Andrei Arion, LesFurets.com, tp-bigdata@lesfurets.com

Plan

1. **GDELT** dataset
2. Projet: Etudier l'évolution de la pandémie COVID19 via son impact media
3. TP (en autonomie): exploration des données GDELT (local ou sur AWS)

GDELT

" The Global Database of Events, Language, and Tone monitors the world's broadcast, print, and web news from nearly every corner of every country in over 100 languages and identifies the people, locations, organizations, themes, sources, emotions, counts, quotes, images and events driving our global society every second of every day, creating a free open platform for computing on the entire world.

Jeu de données GDELT

- trois types de fichiers CSV compresses:
 - les **events** ([schema](#), [CAMEO Ontology](#), [documentation](#))
 - les **mentions** ([schema](#), [documentation](#))
 - le **graph des relations** \Rightarrow GKG, Global Knowledge Graph ([schema](#), [documentation](#))

Index des fichiers

- **masterfilelist.txt** [Master CSV Data File List – English](#)
- **masterfilelist-translation.txt** [Master CSV Data File List – GDELT Translingual](#)

Disponibilite des donnees

- access libre sur <http://data.gdeltproject.org/gdeltv2/>
- disponible egalement sur [Google BigQuery](#)
 - pratique pour explorer la structure des donnees, generation des types de donnees, donees annexes

Projet: Etudier l'évolution de la pandémie COVID19 via son impact media

Objectif: Proposer un **système de stockage distribué, résilient et performant**
sur **AWS** pour les données de GDELT.

Fonctionnalités

1. afficher le nombre d'articles/événements qui parlent de COVID qu'il y a eu pour chaque triplet (jour, pays de l'évènement, langue de l'article)
2. pour un pays donné en paramètre, affichez les événements qui y ont eu place triées par le nombre de mentions (tri décroissant); permettez une agrégation par jour/mois/année
3. pour une source de données passée en paramètre (gkg.SourceCommonName) affichez les thèmes, personnes, lieux dont les articles de cette sources parlent ainsi que le le nombre d'articles et le ton moyen des articles (pour chaque thème/personne/lieu); permettez une agrégation par jour/mois/année.
4. est-ce qu'on observe des patterns dans l'évolution qui pourraient nous permettre d'identifier la prochaine vague/pandemie

C. Contraintes

1. **au moins 1 technologie vue en cours** en expliquant les raisons de votre choix
2. **système distribué et tolérant aux pannes**
3. **une année de données**
4. utiliser **AWS** pour déployer le cluster.

D. Les livrables

- **code source** (lien sur github...)
- **presentation:** *architecture, modélisation, les avantages et inconvénients, des choix de modélisation et d'architecture, volumetrie, limites et contraintes*

Notation

- qualité et clarte de presentation (5/20)
- infra/performances/budget (5/20)
- implementation des fonctionnalités (modelisation/stoquage/requetage) (10/20)

F. Organisation

- travail en equipe (5 personnes)
- **soutenance**
 - Présentation: 10 minutes
 - Démo: 10 minutes
 - Questions & Réponses : 10 minutes

F. Demo

- les données devront être préalablement chargées dans votre cluster
- demo des fonctionnalités
- démontrer la resilience de votre systeme (**chaos monkey**)

Questions ?

TP: Exploration des donnees GDELT

- continuer l'*exploration libre*
 - sur **AWS** voir [TP AWS](#)
 - ou en **local**: Zeppelin 0.8.0 + Spark 2.3.2 (linux/match EMR versions!)
 - i7/SSD/32GB RAM ⇒ 20 jours d'événements (~30GB compressées US+translations)
- exemples de code "*pedagogiques*"
- questions/erreurs

Approche:

1. explorer les **donnees**: *types/organisation/volumetrie*
2. explorer les **fonctionalitees** demandees
3. identifier les **agregats**, comment les stocker
 1. \Rightarrow decider la **techno/modelisation**
 2. tests/ajustements/optimisations

Disponibilite des donnees

1. telechargement depuis <http://data.gdeltproject.org/>
 - stockage sur S3
 - stockage en local

Script de telechargement

```
def fileDownloader(urlOfFileToDownload: String, fileName: String) = {  
  val url = new URL(urlOfFileToDownload)  
  val connection = url.openConnection().asInstanceOf[HttpURLConnection]  
  connection.setConnectTimeout(5000)  
  connection.setReadTimeout(5000)  
  connection.connect()  
  
  if (connection.getResponseCode >= 400)  
    println("error")  
  else  
    url #> new File(fileName) !!  
}  
  
fileDownloader("http://data.gdeltproject.org/gdeltv2/masterfilelist.txt",  
  "/home/aar/bigdata/proj2018/data/masterfilelist.txt") // save the list file to the Spark Maste
```


Stockage S3

```
import com.amazonaws.services.s3.AmazonS3Client
import com.amazonaws.auth.BasicAWSCredentials

val AWS_ID = "*****"
val AWS_KEY = "*****"
val awsClient = new AmazonS3Client(new BasicAWSCredentials(AWS_ID, AWS_KEY))

sc.hadoopConfiguration.set("fs.s3a.access.key", AWS_ID) //(1) mettre votre ID du fichier creden
sc.hadoopConfiguration.set("fs.s3a.secret.key", AWS_KEY) //(2) mettre votre secret du fichier c
sc.hadoopConfiguration.set("fs.s3a.connection.maximum","1000") //(3) 15 par default !!!

awsClient.putObject("john-doe-telecom-gdelt2018", "masterfilelist-translation.txt",
new File( "/mnt/tmp/masterfilelist-translation.txt") )
```


Stockage (parallel)

```
sqlContext.read.  
  option("delimiter"," ").  
  option("infer_schema","true").  
  csv("/tmp/data/masterfilelist-translation.txt").  
  withColumnRenamed("_c2","url").  
  filter(col("url").contains("/201812")).  
  repartition(200).foreach( r=> {  
    val URL = r.getAs[String](0)  
    val fileName = r.getAs[String](0).split("/").last  
    val dir = "/home/aar/bigdata/proj2018/data/"  
    val localFileName = dir + fileName  
    fileDownloader(URL, localFileName)  
    val localFile = new File(localFileName)  
    /* AwsClient.s3.putObject("john-doe-telecom-gdelt2018", fileName, localFile )  
       localFile.delete() */  
  })
```


Chargement events

```
val eventsRDD = sc.binaryFiles("../data/20181[0-9]*.export.CSV.zip",100).
  flatMap { // decompresser les fichiers
    case (name: String, content: PortableDataStream) =>
      val zis = new ZipInputStream(content.open)
      Stream.continually(zis.getNextEntry).
        takeWhile(_ != null).
        flatMap { _ =>
          val br = new BufferedReader(new InputStreamReader(zis))
          Stream.continually(br.readLine()).takeWhile(_ != null)
        }
  }
val cachedEvents = eventsRDD.cache // RDD
```

```
cachedEvents.take(1)
```

```
res5: Array[String] = Array(807502237    20171204          201712   2017      2017.9151
```


RDD / DataSets / SparkSQL

Use types stored in BigQuery

```
~/b/p/schema >>> google-cloud-sdk/bin/bq show --format=prettyjson gdelt-bq:gdeltv2.events  
  
{  
  "creationTime": "1463679409113",  
  "etag": "O5/wP5Yg/W1CN+rDMM85/w==",  
  "id": "gdelt-bq:gdeltv2.events",  
  "kind": "bigquery#table",  
  "lastModifiedTime": "1545240340088",  
  "location": "US",  
  "numBytes": "171542041673",  
  "numLongTermBytes": "0",  
  "numRows": "399903677",  
  "schema": {  
    "fields": [  
      {  
        "description": "Globally unique identifier assigned to each event record that uniquely
```


BigQuery JSON Type \Rightarrow case class Event

```
~/b/p/schema >>> google-cloud-sdk/bin/bq show --format=prettyjson gdelt-bq:gdeltv2.events |  
jq '["schema"]' events.json |jq '[]|.[]|.name+": " + .type + ","|.[]' |sed "s/INTEGER/  
sed "s/FLOAT/Double/" | sed "s/STRING/String/" |sed "s/\"/"/g"
```


Event case class

```
case class Event(  
  GLOBALEVENTID: Int,  
  SQLDATE: Int,  
  MonthYear: Int,  
  Year: Int,  
  FractionDate: Double,  
  Actor1Code: String,  
  Actor1Name: String,  
  Actor1CountryCode: String,  
  Actor1KnownGroupCode: String,  
  Actor1EthnicCode: String,  
  Actor1Religion1Code: String,  
  Actor1Religion2Code: String,  
  Actor1Type1Code: String,  
  Actor1Type2Code: String,  
  Actor1Type3Code: String
```


Data cleanup (wrangling/munging)

- correction des types de donnees (STRING→DATE,INT→BIGINT...)
- correction des valeurs (lignes incompletes, valeurs incorrectes...)
- ...

RDD \Rightarrow DataSet \Rightarrow SparkSQL

```
def toDouble(s : String): Double = if (s.isEmpty) 0 else s.toDouble
def toInt(s : String): Int = if (s.isEmpty) 0 else s.toInt
def toBigInt(s : String): BigInt = if (s.isEmpty) BigInt(0) else BigInt(s)

cachedEvents.map(_.split("\t")).filter(_.length==61).map(
  e=> Event(
    toInt(e(0)),toInt(e(1)),toInt(e(2)),toInt(e(3)),toDouble(e(4)),e(5),e(6),e(7),e(8),e(9)
    e(21),e(22),e(23),e(24),toInt(e(25)),e(26),e(27),e(28),toInt(e(29)),toDouble(e(30)),toI
    toDouble(e(41)),e(42),toInt(e(43)),e(44),e(45),e(46),e(47),toDouble(e(48)),toDouble(e(4
  )).toDS.createOrReplaceTempView("export")
  spark.catalog.cacheTable("export")
```


Exploration via Zeppelin/SparkUI

```
z.show(spark.sql("""
  SELECT *
  FROM export
  """))
```

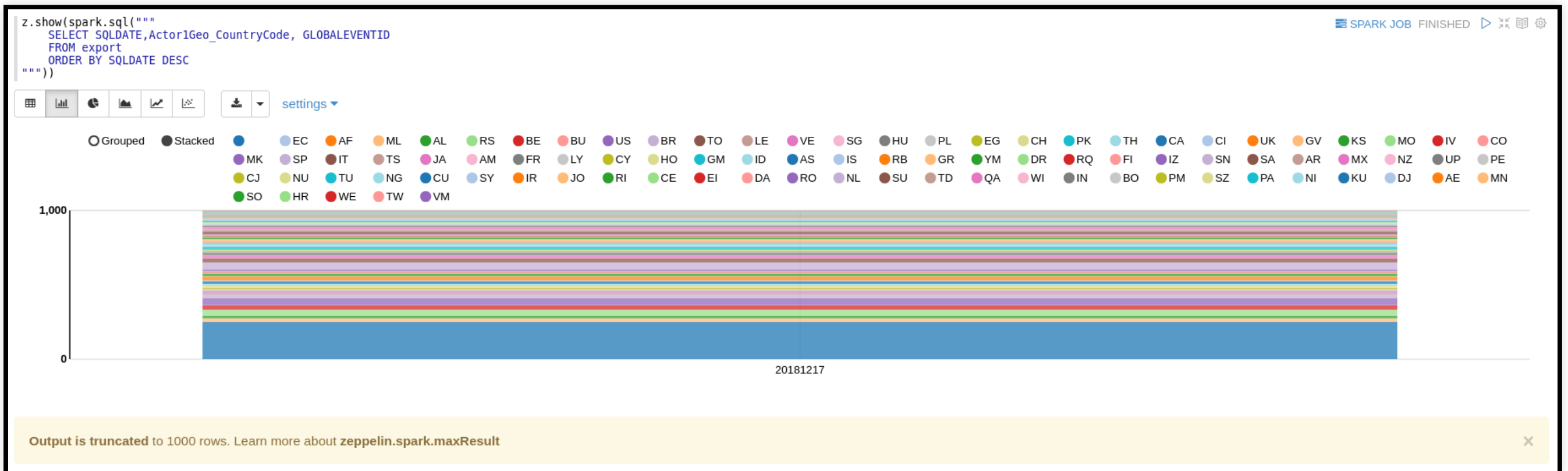
settings ▾

GLOBAL EVENT ID ▾	SQL DATE ▾	Month Year ▾	Year ▾	Fraction Date ▾	Actor 1 Code ▾	Actor 1 Name ▾	Actor 1 Country Code ▾
807502255	20181104	201811	2018	2018.8329			
807502256	20181104	201811	2018	2018.8329			
807502257	20181104	201811	2018	2018.8329			
807502264	20181127	201811	2018	2018.8959			
807502265	20181127	201811	2018	2018.8959			
807502266	20181127	201811	2018	2018.8959			
807502292	20181203	201812	2018	2018.9123			
807502296	20181204	201812	2018	2018.9151			

Output is truncated to 102400 bytes. Learn more about ZEPPELIN_INTERPRETER_OUTPUT_LIMIT

Took 1 min 26 sec. Last updated by anonymous at December 19 2018, 4:15:42 PM. (outdated)

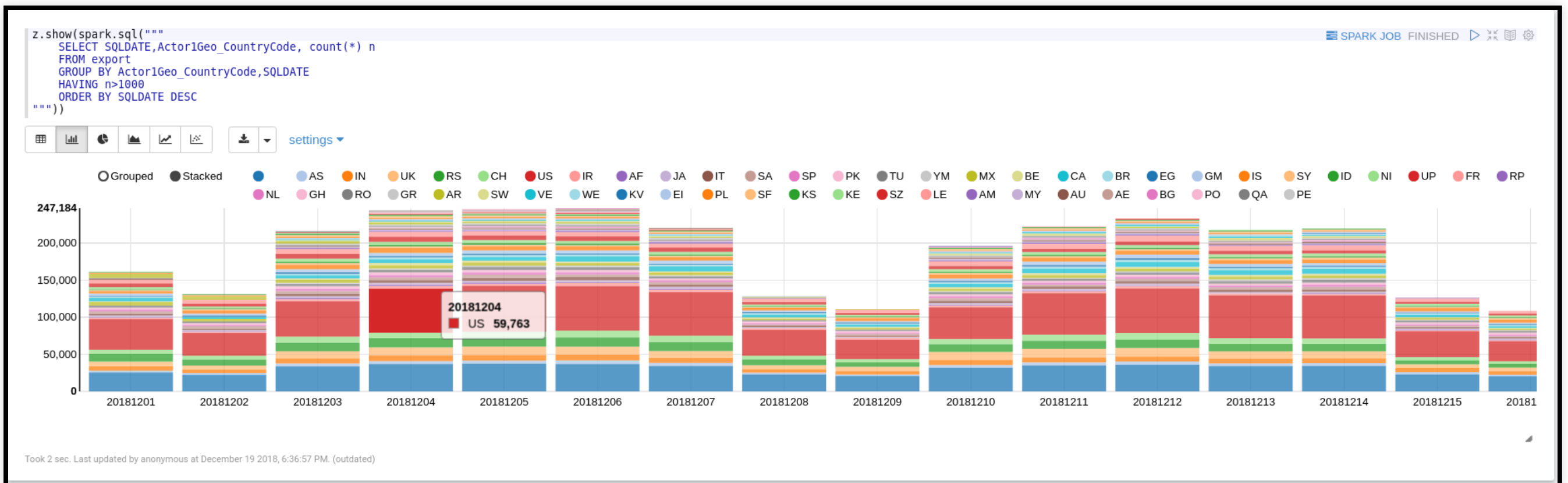
Output is truncated



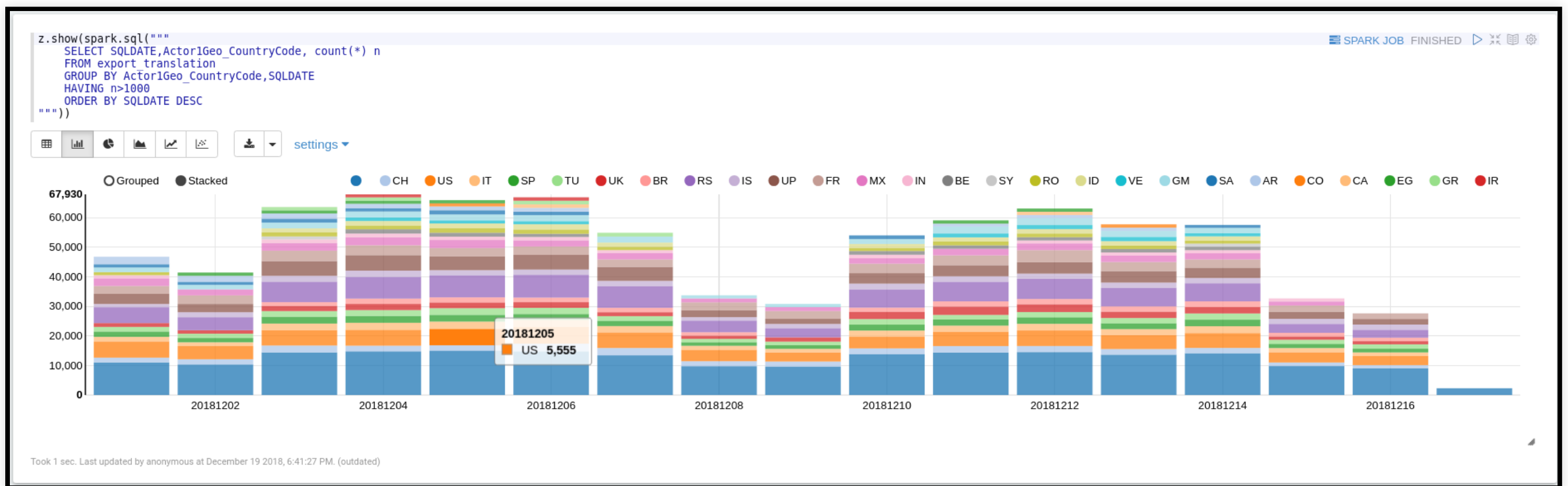
Volumetrie




Events by Actor1Country (export)



Events by Actor1Country (export-translation)



Volumetrie (cached table)

 2.3.2

Zeppelin application UI

JobsStagesStorageEnvironmentExecutorsSQL

Storage

RDDs

ID	RDD Name	Storage Level	Cached Partitions	Fraction Cached	Size in Memory	Size on Disk
7	In-memory table export	Disk Serialized 1x Replicated	2	100%	0.0 B	715.8 MB
32	In-memory table export_translation	Disk Serialized 1x Replicated	1	100%	0.0 B	768.9 MB

Volumetrie (cached table)

