Crisis_analyse

August 14, 2019

Intern

1 Interns project for world crisis analysis

In this notebook we are going to present you the work we achieved here in the **AD Center** of **Capgemini** ! We have been throught different tools, softwares and work methods in order to develop a usefull and powerfull tool to manipulate **BIG DATA**. *But what exactly is the idea ?* We wanted to find a way to analyse the crisis evolution in each country of the world with respect to officials datas taken from well-known websites To reach this goal, we used **3** different main tools : 1. Spark and its pyspark librairy as we wanted to use Python 2. Docker and its containers 3. Jenkins and its pipeline Let's take a look at what we are manipulating now !

1.0.1 With which kind of data we are playing with ?

Let's start with the begining and see what are the **3 data bases** we are using for our analysis :

1. A Json Object taking is data directly from the ***Food and Agriculture Organization*** database.In this file we can find informations about the agricultural production of each country in the world from **1960** to **2018**

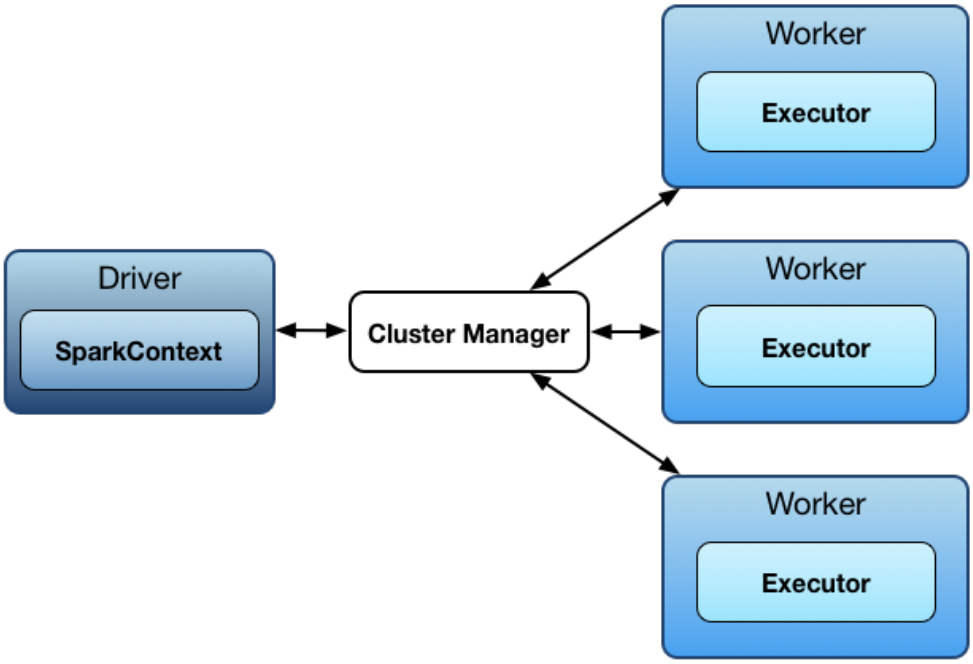
2. A CSV file that can be read with Excel and which takes is data directly from the famous ***WorldBank*** website. In this file we can find the amount of population for each country in the world from **1960** to **2018**

3. A DB file which is literally a SQL data base that can be read with DB Browser and which takes is data from ***WorldBank*** website too. In this file we can find the GDP/PIB and a growth indicator for each country in the world from **1960** to **2018**

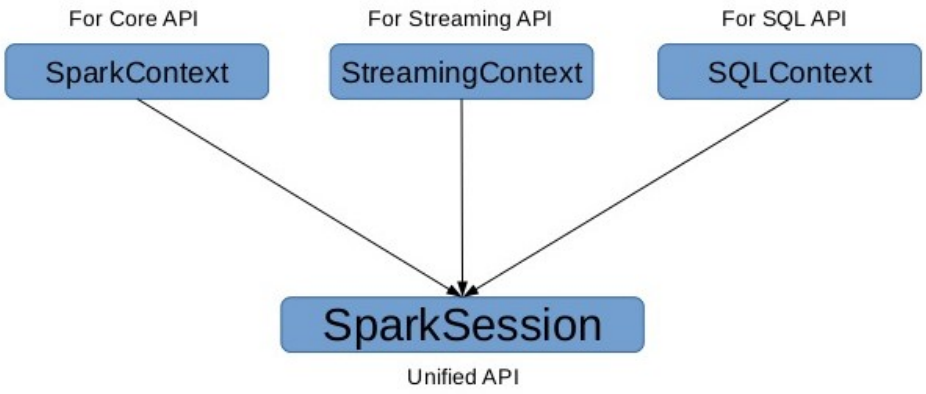
1.0.2 Benefits of using Spark librairies and architecture

Spark is a powerfull and open-source framework in order to manage big data. As we use Python, we used the library pyspark to play with the data we found on the internet. But let's see what's bring us to use Spark Here we got all the advantages of **Spark** in one scheme :

These advantages are due to a special architecture :

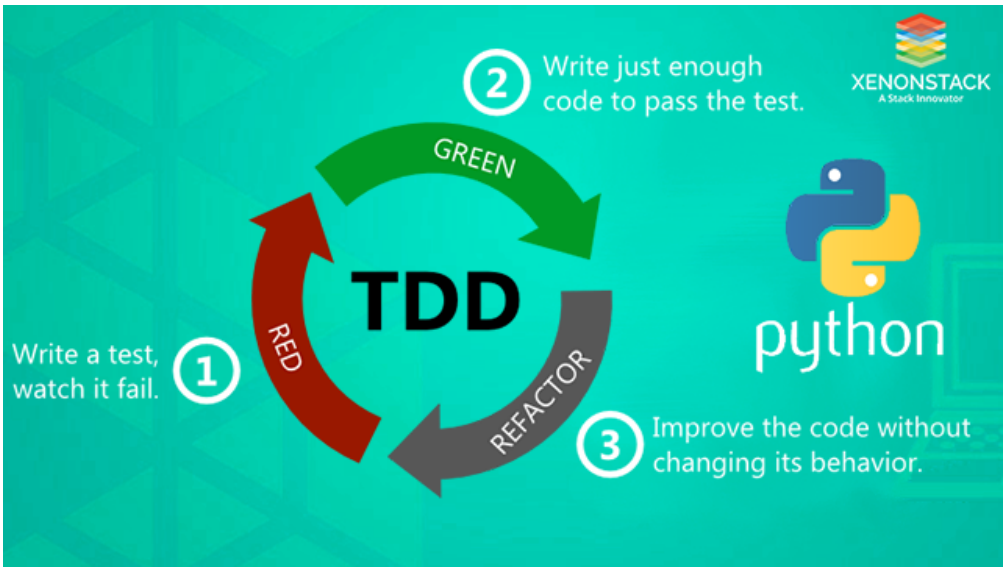


Furthermore, the librairies of Spark allowed us to create the most import and usefull things we needed to start to play with the project: the **SparkSession** : It allows you to use independently the **SparkContext**, **StreamingContext** or **SQLContext**



1.0.3 TDD Method

Another method which has been really important for our project is The TDD method. It has been really usefull in order to developp the test of our different functions



Intern

1.0.4 Jupyter: great notework for developer and data scientist

Jupyter is a powerful web Application used as an IDE. This tool is very usefull for data analysis projects. Comparing to other IDEs, it offers the following benefits:

- Interactive results : with cells, jupyter allows to test each part of the code instead of the whole code. Then we can visualize datas and see the results without having to leave the environment or to run the entire program.
- Supports more than 40 differents programming languages

- Trying various perspectives : with cells, we can code differents programs/fonctions independently. Then it moves easier to investigate other perspective without making a new file.
- Better gestion of teamword
- Web Application

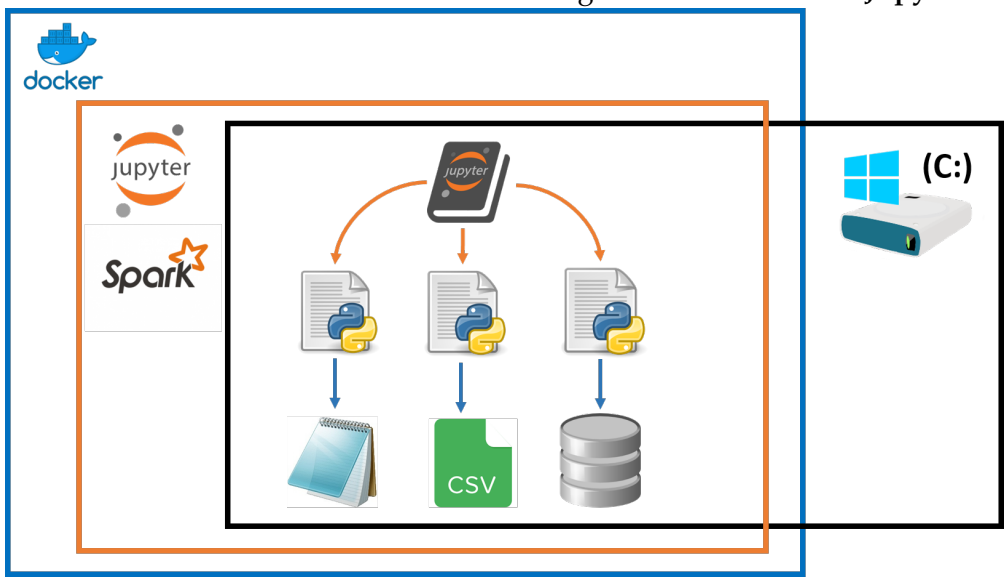
1.0.5 Bonus :

How to execute Jupyter Notebook from prompt ?

pip install nbval
py.test –nbval my_notebook.ipynb

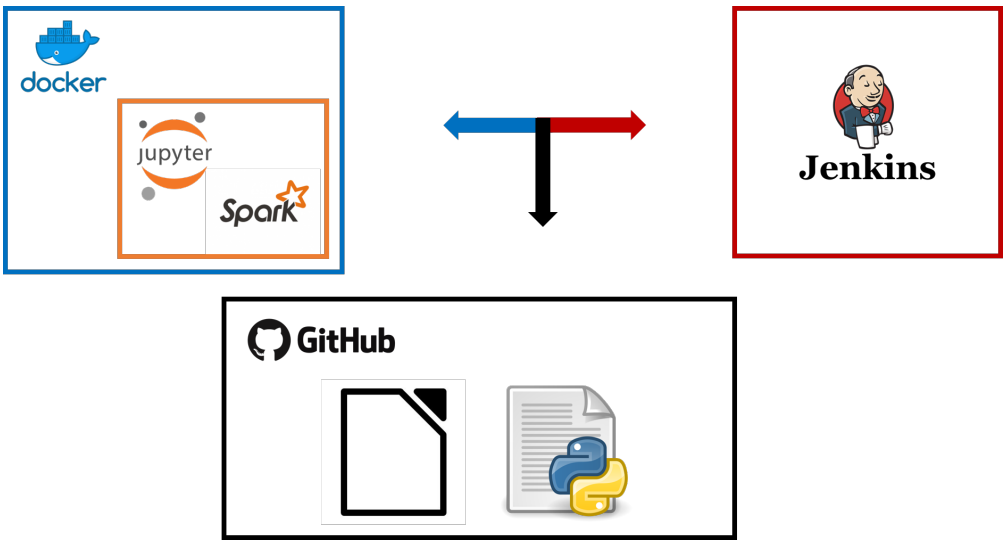
1.0.6 Project workspace and environnements in local

This is a representation of how we configured the local workspace of the project. There are **three common environnements** that we tried to run together : 1. **Docker** 2. **Jupyter - Spark** 3. **Local Disc**



1.0.7 Continuous Integration using Docker, Jenkins and Pyspark

This is the most interesting part of the project. Indeed, we tried to **apply the Continuous Integration on how we were managing the project in local**. We finally found a apropriate architecture :



Intern

1.0.8 Now let's start to play with the data bases !

We are going here to show you the direct applications of our functions and results throught various schemes and graphics. However we are still going to pass throught some lines of code...as it is important to grab the idea of the entire process !

1.0.9 Importing the requirements

Here some import to allow us to work with pyspark and plotly, which last one is a powerfull library to plot graphics and schemes

[1]:
```
from pyspark import SparkContext
from pyspark import SQLContext
from operator import add
from pyspark.sql import SparkSession
import matplotlib.pyplot as plt
import plotly.graph_objects as go
```

Other imports ! But here, it is in order to call our different classes in which we have all our functions defined

[2]:
```
import sys
sys.path.append('../FAO')
from FAO_production import FAO_production
sys.path.append('../WorldGDP')
from SQLiteNotebook import Spark_GDP
sys.path.append('../employment')
from Unemployment import Employment
```

[3]:
```
spark = SparkSession.builder.appName("Crisis_analyse").getOrCreate()
```

1.0.10 Playing with the JSON data base

Here doing the connexion with the data base

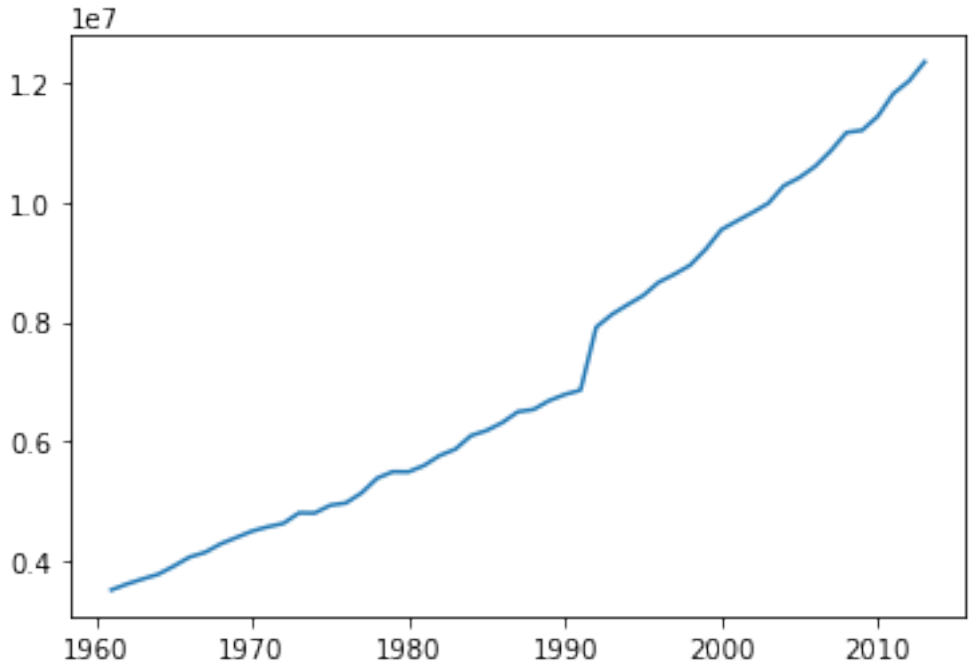
[4]:
```
datapth = "../data2/FAO+database.json"
sqlCtx = SQLContext(spark.sparkContext)
data = sqlCtx.read.json(datapth, multiLine=True)
```

Just a little instance of the first class

[5]:
```
FAO = FAO_production(data, spark)
```

Applying sum function to gradualy see the raise of the **world production throught the years** :

[6]:
```
FAO.plot_production_sum()
```



Another graph but here to see the production with rescpect to different geographic areas :

[7]:
```
FAO.plot_zones_production()
```

This time it's a little bit more funny, you can see a worldmap with a **color indicator** for each country. This color indicator represents the level of food production of the country within the year submited in the function's call :

[8]:
```
FAO.plot_production_world(2008)
```

1.0.11 Playing with the SQLite data base

Here creating the instance of the database

[9]:
```
import sqlite3
```

[10]:
```
datapath = "../data2/world-gdp.db"
sqlitedata = sqlite3.connect(datapath)
```

[11]:
```
GDP = Spark_GDP(sqlitedata, spark)
```

Playing here with the database

[12]:
```
pandaDF = GDP.createGDPPandaDF()
```

[13]:
```
sparkDF = GDP.createSparkDFfromPandaDF(pandaDF)
```

[14]:
```
sparkDF_filtered = GDP.replaceNullValuesTo0fromSparkDF(sparkDF, "all", ["gdp",␣
↪"growth"])
```

Choosing the **range of years** here

[15]:
```
year_frame = [2009,2012]
```

[16]:
```
CountryGDP_panda_DF = GDP.PandaDFOfGDPperCountryfromSparkDF(sparkDF_filtered,␣
↪year_frame)
```

[17]:
```
GDP.plotLinesGrowthwithPandaDF(CountryGDP_panda_DF,year_frame)
```

1.0.12 Playing with CSV database to calculate unemployment

Here creating the instance of the database

[18]:
```
datapath = '../data2/Book2.csv'
data = spark.read.format('csv').options(header='true', inferSchema='true',␣
↪delimiter=';').load(datapath)
```

[19]:
```
Unemployment = Employment(data, spark)
```

[20]:
```
geo_zone = {'United States':'US', 'European Union': 'Europe', \
            'Singapore':'Asian Tigers', 'Korea, Rep.':'Asian Tigers' , 'Hong␣
↪Kong':'Asian Tigers', \
             'Brazil':'BRICS', 'Russian Federation':'BRICS', 'India':'BRICS',␣
↪'South Africa':'BRICS'}
```

[21]:
```
years_frame = [2004,2014]
```

[22]:
```
Frame_studied = Unemployment.extract_info(years_frame,['United States',␣
↪'European Union', 'China', 'Japan', \
                            'Singapore', 'Korea, Rep.', 'Hong Kong' \
                            'Brazil', 'Russian Federation', 'India', 'South␣
↪Africa'])
```

[26]:
```
Frame_studied_clased = Unemployment.add_groups(Frame_studied, geo_zone)
```

[27]:
```
dataframeSpark = Unemployment.groups_data(Frame_studied_clased,years_frame)
```

[28]:
```
Unemployment.plot_unemployment(dataframeSpark, years_frame)
```