1. **Give a brief overview of how you integrated each system.**

I launched an EC2 instance (t2.micro) with the EBS volume attached and then mounted:
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-attaching-volume.html
http://docs.aws.amazon.com/AWSEC2/latest/UserGuide/ebs-using-volumes.html

At the time data were available, I dig into the information and realized only files under *"/aviation/airline_ontime"* were required to complete all the question along the different groups. The imported fields were: (*Year, Month, DayofMonth, Weekday, UniqueCarrier, FlightNum, Origin, Dest, CRSDepTime, DepDelay, CRSArrtime, ArrDelay, Cancelled*).

A python script was developed to extract, clean and import data to "airlineontime" S3 bucket. Public and secret key were provided to EC2 instance, with a proper policy providing rights to load data into S3 bucket. Different python modules were used to complete every step:

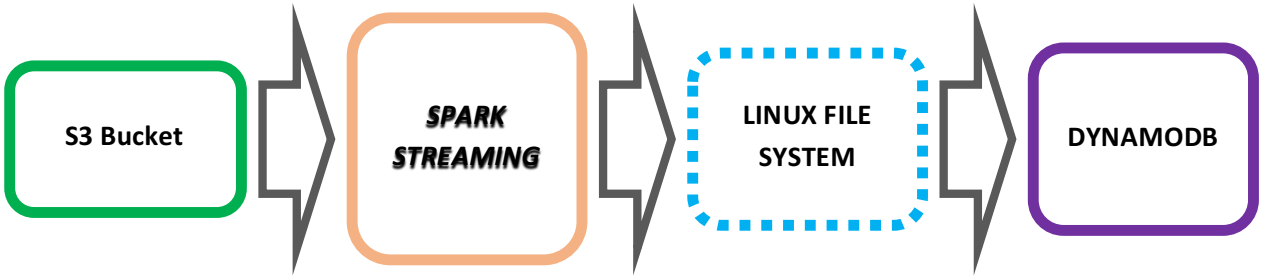
- The zipfile module was used to manipulate ZIP archive files.
- The os module was used to manipulate path and read all the lines in all the files.
- The csv module was used to manipulate all the files in csv format.
- The Boto module is the AWS SDK for Python, which allows write software that makes use of Amazon services like S3 EC2 and DynamoDB.

At this point of time data from S3 were ready to be streamed by Spark, a second python script was developed to provide an input stream from "*airlineontime*" S3 bucket. Additionally, to the python modules previously defined, two more modules were required:

- The socket module was used to provide a TCP socket endpoint for spark streaming logic to get an input stream.
- The key module from boto.s3.key was required to expose data from S3 bucket to spark streaming applications.

Spark streaming applications, coded in python, connect to input stream by means of *socketTextStream* function provided by *StreamingContext* contained in *pyspark.*streaming module. *SocketTextStream* create an input from TCP source *hostname:port*. Data is received using a TCP socket and received byte is interpreted as UTF8 encoded\n delimited lines. Following modules were used to process the data stream: *SparkContext, SparkConf and StreamingContext.*

Streamed data is processed by Spark Streaming applications, which make use of Boto module, as explained, and the output is stored in Linux File System or loaded into DynamoDB tables, depending on requirements.



2. **What approaches and algorithms did you use to answer each question in each system?**

Results are required for the complete set of data contained in the stream, nevertheless small batches of input data are used for Spark Engine as process unit. Batch processing needs to be stateful in most of the exercises and the way to achieve this requirement is by means of *updateStateByKey* DStreams transformation and RDD checkpointing:

streamingContext.checkpoint('path')

Following piece of code shows how DStream is filtered (group 1 exercise 3 example). These approach has been followed along the different exercises:

```
filtered = lines.map(lambda line: line.split(","))\
                .map(lambda flight: (flight[4], (flight[11], 1)))\
                .updateStateByKey(updateFunction)
```

The state of each key is updated by applying *updateFunction* on the previous state of the key and the new values for the key.

Depending on the nature of the exercises key contains one or two values:

- Group 1 ex.1 and group 3 ex.1: keys; origin and destination for two different streams.
- Group 1 exercise 2: key; carrier.
- Group 1 exercise 3: key; dayofweek.
- Group 2 exercise 1: key; origin, carrier.
- Group 2 exercises 2 and 4: key; origin, destination.
- Group 3 exercise 2: follows a different approach based on a two stream properly filtered and joined using as key date-flightY. Results are stored in DynamoDB per batch if there is no a better route stored yet.

To load data into DynamoDB tables, the dynamodb connector provided by python boto module was used, specifically the modules dynamodb2, Table and Item.

```
dynamodb_conn=
boto.connect_dynamodb(aws_access_key_id='MY_ACCESS_KEY_ID',
aws_secret_access_key='MY_SECRET_ACCESS_KEY')

table = Table(DBTable, dynamodb_conn)

item = Item(table, data)

item.save(overwrite=True)
```

3. What are the results of each question?

Group 1 (Answer any 2):

1. Rank the top 10 most popular airports by numbers of flights to/from the airport.

airport	*total*
ORD	*12051796*
ATL	*11323515*
DFW	*10591818*
LAX	*7586304*
PHX	*6505078*
DEN	*6183518*
DTW	*5504120*
IAH	*5416653*
MSP	*5087036*
SFO	*5062339*

2. Rank the top 10 airlines by on-time arrival performance.

carrier	*mean_delay*
HA	*-1.01180434575*
AQ	*1.15692344248*
PS	*1.45063851278*
ML (1)	*4.74760919573*
PA (1)	*5.32243099993*
F9	*5.46588114882*
NW	*5.55778339267*
WN	*5.56077425988*
OO	*5.73631246366*
9E	*5.8671846617*

3. Rank the days of the week by on-time arrival performance.

weekday	*mean_delay*
6	*4.30166992608*
2	*5.99045884132*
7	*6.61328029244*
1	*6.71610280259*
3	*7.20365639467*
4	*9.09444100834*
5	*9.72103233759*

Group 2 (Answer any 3):

1. For each airport X, rank the top-10 carriers in decreasing order of on-time departure performance from X.

Airport	*Carrier-Mean_delay*
SRQ	*TZ-0,XE-1,AA-3,UA-3,US-3,YV-3,DL-4,NW-4,TW-4,FL-5*
CMH	*AA-3,DH-3,DL-4,ML(1)-4,NW-4,EA-5,PI-5,US-5,TW-6,YV-7*
JFK	*UA-5,CO-8,DH-8,XE-8,AA-10,B6-11,DL-11,NW-11,PA(1)-11,MQ-12,TW-12*
SEA	*OO-2,PS-4,YV-5,AA-6,DL-6,HA-6,NW-6,TZ-6,US-6,CO-7,EV-7,F9-7*
BOS	*TZ-3,PA(1)-4,ML(1)-5,DL-7,EV-7,NW-7,AA-8,EA-8,US-8,XE-8*

2. For each airport X, rank the top-10 airports in decreasing order of on-time departure performance from X.

airport	*List of top-10 airports*
SRQ	*EYW,TPA,MEM,IAH,RDU,MDW,MCO,FLL,BNA,CLE*
CMH	*OMA,SYR,AUS,SDF,CLE,MSN,CAK,SLC,IAD,DFW*
JFK	*SWF,ANC,ISP,ABQ,MYR,UCA,BGR,BQN,STT,CHS,*
SEA	*EUG,PIH,PSC,CVG,MEM,YKM,SNA,LIH,IAH,DTW*
BOS	*SWF,ONT,AUS,GGG,MSY,LGA,OAK,LGB,MDW,BDL*

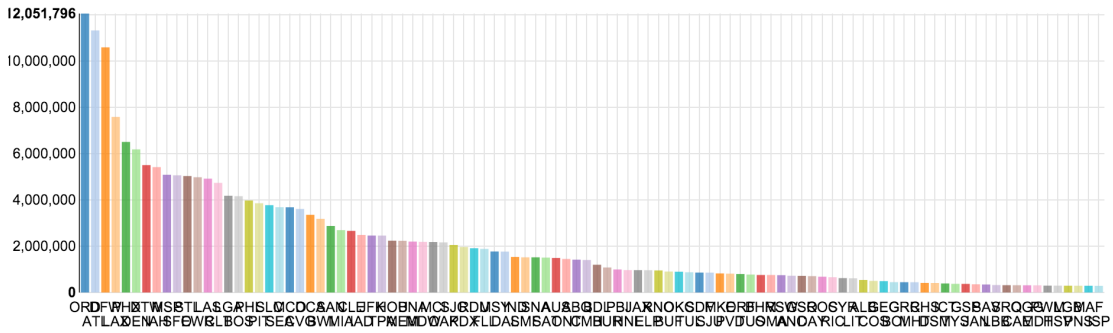
3. For each source-destination pair X-Y, rank the top-10 carriers in decreasing order of on-time arrival performance at Y from X.
4. For each source-destination pair X-Y, determine the mean arrival delay (in minutes) for a flight from X to Y.

origin	*Destination*	*mean_delay*
LGA	*BOS*	*1*
BOS	*LGA*	*3*
OKC	*DFW*	*4*
MSP	*ATL*	*6*

Group 3 (Answer both using only Hadoop and Spark):

1. Does the popularity distribution of airports follow a Zipf distribution? If not, what distribution does it follow?

Do not follow a Zipf distribution, on the other hand, log-normal seems the best at modelling the data.



2. Find, for each X-Y-Z and day/month combination in the year 2008, the two flights (X-Y and Y-Z) that satisfy constraints, if such flights exist.

route	***depdate***	***flight_xy***	***flight_yz***
BOS-ATL-LAX	*2008-04-03*	*FL270*	*DL75*
PHX-JFK-MSP	*2008-09-07*	*B6178*	*NW609*
DFW-STL-ORD	*2008-01-24*	*AA1336*	*AA2245*
LAX-MIA-LAX	*2008-05-16*	*AA280*	*AA456*

4. **What system-level or application-level optimizations (if any) did you employ?**
   DynamoDB: sort keys were properly selected in order to obtain results faster.
   S3 storage: buckets provide persistent storage which reduce wastes populating HDFS every time the EMR cluster is stopped.
   Spark streaming: use of transformations on DStreams instead of running SQL programmatically, which is very inefficient, using SQLContext and Dataframes,

5. **Your opinion about whether the results make sense and are useful in any way.**
   Conclusion about size/popularity of different airport was quite predictable. The information obtained regarding different types of mean delay were not predictable at all and are quite useful from different points of views: carriers could take actions to reduce delays just in case these are not acceptable and passengers could adapt their preferences not only about carriers but also regarding flights' scheduling to avoid delays.

6. **How did the different stacks (Hadoop and Spark) from Task 1 and Task 2 compare? Which stack did you find the easiest to use? The fastest?**
   Hadoop is extremely easy of use by means of HIVE queries, on the other hand, can be very tedious to code map/reduce functions, in many different scenarios, by yourself.
   Spark streaming can be ease of use running SQL programmatically but very inefficient and can be a little more complicated to code but run very fast using input and output transformations on DStreams.

   VIDEO DEMONSTRATION:
   http://sendvid.com/m06ah0qr