The goal of this task is to process streaming data. In the Task 1 we were asked to work on the dataset from the US Bureau of Transportation Statistics (BTS). The dataset was hosted as an Amazon EBS volume snapshot with US Snapshot ID (Linux/Unix): snap-e1608d88 and size of 15 GB. Therefore, I have decided to utilize the AWS-EC2 to copy and extract the data. After extracting the data and selecting the necessary fields I have assembled all the data from different files to a single csv file and store it into AWS-S3 and performed the requirements for the Task 1 of the project. In the task 2 we are asked to somehow ingest the above data as a real-time live data streaming and then perform real time process streaming.

1. Overview of how the system is integrated.

For my data streaming I have used AWS-Kinesis and my stream processing system is Spark streaming. My integration for the task 2 of the project is as follow:

First we have been asked to ingest data into the streaming system. To do so:

1. I have created an AWS-EC2 instance. EC2 instance: (Amazon Linux 2 AMI) (t2.medium) (SSD 20 GB).
2. Creating an appropriate IAM to connect the EC2 instance into AWS-S3 and AWS-Kinesis and inset keys using IAM configuration and set the key in the instance using:

   aws configure
3. From Task-1 of the project I have stored the cleaned data assembled as a single CSV file in my S3 bucket. I copied the cleaned data from my S3 bucket into the EC2 instance:

   aws s3 cp s3://cloudcoursecap/aviation/CleanedData.csv .
4. To ingest the data as a streaming data first I have created an AWS-kinesis stream with 100 shard.
5. From the EC2 instance using boto3 library in python I send the data as streaming data package to the AWS-kinesis stream using the following python code:

```python
# -*- coding: utf-8 -*-
"""Created on Wed Jul 22 05:54:52 2020 @author: Bahman"""
import time
import datetime
import pandas as pd
import pan as pd
from boto import kinesis
from datetimport kinesis
from datetime import timedelta
if __name__ == '__main__':
    kinesis = kinesis.connect_to_region("us-east-1")
    col_names =  ['Year','Month','Origin','ArrTime','DepTime','Dest','ArrDelay',
            'UniqueCarrier','DayOfWeek','DayofMonth','DepDelay','Cancelled','timeStamp']

    packageSize = 2000

    while True:
        fp = open("CleanedData.csv")
        reader = csv.reader(fp)
        dataPackage = []
        count = 0
        next(reader)
        for row in reader:
            if count < packageSize:
                count = count + 1
                timer = datetime.now().strftime("%H_%M")
                row = row + [timer]
                dataPackage.append(row)
            else:
                date = datetime.today().strftime("%Y_%m_%d")
                timer = datetime.now().strftime("%H_%M")
                browser_df = pd.DataFrame(dataPackage, columns=col_names)
                data = bytes(browser_df.to_json(orient='records'))
                #print(data)
                kinesis.put_record("bahmanStream", data, "Accesslog")
                print("Data has been sent!")
                time.sleep(1)
                dataPackage = []
                count = 0
        fp.close()
```

Fig. 1. Python code to ingest the data into kinesis stream.

6. To check the streaming data I have also created a Kinesis analytics application and connected it to my Kinesis stream. With the Kinesis analytics application we can check the data schema and use SQL for live data inquiry.
7. I have also created a Kinesis stream delivery and connected it to a S3 bucket so that stream data are timestamped and stored as streaming data.
8. Create an AWS-EMR (elastic MapReduce):
   - I have used **EMR** 6.0.0 with **Hadoop** 3.2.1 and **Spark** 2.4.4
   - I have also used the **bootstrap** option to install necessary libraries and java files.
9. After connecting to kinesis streamed data I utilized PySpak SQL to inquiry and then I have used writeStream to show them on the concole.
10. For the questions which needed to be stored on **DynamoDB** first I have created a separate table for each question in DynamoDB.
11. To efficiently store the queries results into the DynamoDB tables I have created an AWS **Lambda** function to automatically push the data to DynamoDB table as explained below.
12. I set the trigger for the Lambda function as creation of a file on a S3 bucket using **boto3**. For example here is the Lambda function code for question3-2:

```
# -*- coding: utf-8 -*-
"""Created on Sun Jul  23 17:20:44 2020 @author: Bahman"""
import boto3
s3 = boto3.client('s3')
dynamodb = boto3.resource('dynamodb')

def csv_reader(event, context):
    bucket = event['Records'][0]['s3']['bucket']['name']
    key = event['Records'][0]['s3']['object']['key']
    obj = s3.get_object(Bucket=bucket, Key=key)
    rows = obj['Body'].read().split('\n')
    table = dynamodb.Table('group3_2')
    Origin_X = []; DayofMonth_X = {};Month_X = {};DepTime_X = {};Dest_X = {}
    ArrDelay_X = {};DayofMonth_Y = {};Month_Y = {};DepTime_Y = {};Dest_Y = {}
    ArrDelay_Y = {};TotalArrivalDelay = {}
    with table.batch_writer() as batch:
        for row in rows:
            splited = row.split(',')
            if len(splited) < 2:
                continue
            if splited[3] in Origin_X:
                DayofMonth_X[splited[3]].append(splited[0])
                Month_X[splited[3]].append(splited[1])
                DepTime_X[splited[3]].append(splited[2])
                Dest_X[splited[3]].append(splited[4])
                ArrDelay_X[splited[3]].append(splited[5])
                DayofMonth_Y[splited[3]].append(splited[6])
                Month_Y[splited[3]].append(splited[7])
                DepTime_Y[splited[3]].append(splited[8])
                Dest_Y[splited[3]].append(splited[9])
                ArrDelay_Y[splited[3]].append(splited[10])
                TotalArrivalDelay[splited[3]].append(splited[11])
            else:
                Origin_X.append(splited[3])
                DayofMonth_X[splited[3]] = [splited[0]]
                Month_X[splited[3]] = [splited[1]]
                DepTime_X[splited[3]] = [splited[2]]
                Dest_X[splited[3]] = [splited[4]]
                ArrDelay_X[splited[3]] = [splited[5]]
                DayofMonth_Y[splited[3]] = [splited[6]]
                Month_Y[splited[3]] = [splited[7]]
                DepTime_Y[splited[3]] = [splited[8]]
                Dest_Y[splited[3]] = [splited[9]]
                ArrDelay_Y[splited[3]] = [splited[10]]
                TotalArrivalDelay[splited[3]] = [splited[11]]
        for item in Origin_X:
            batch.put_item(Item={
                'Origin_X' : item,
                'DayofMonth_X' : DayofMonth_X[item],
                'Month_X' : Month_X[item],
                'DepTime_X' : DepTime_X[item],
                'Dest_X' : Dest_X[item],
                'ArrDelay_X' : ArrDelay_X[item],
                'DayofMonth_Y' : DayofMonth_Y[item],
                'Month_Y' : Month_Y[item],
                'DepTime_Y' : DepTime_Y[item],
                'Dest_Y' : Dest_Y[item],
                'ArrDelay_Y' : ArrDelay_Y[item],
                'TotalArrivalDelay' : TotalArrivalDelay[item]
            })
```

Fig. 2. Python code in AWS-Lambda function to push the results from S3 to DynamoDB.

13. So when in PySpark I obtain the result for a question, I stored the obtained result into S3 then automatically it will be pushed to the DynamoDB table using the Lambda function. It should be mention that it was challenging for me to how I can saved the live batch quarries into S3. For example for Group 1 question 2 I used this commands to show the results in the console:

```
# 2. Rank the top 10 airlines by on-time arrival performance.
df_count= df.groupBy("UniqueCarrier").avg("ArrDelay").orderBy('avg(ArrDelay)',
                      ascending=True)
df_count= df_count.writeStream.outputMode("complete").format("console").start()
df_count.awaitTermination()
```

So because I used the “complete” mode spark doesn’t allow to save the results as file so to save the results (the last batch) we should store the results in the memory and then send it to S3 as:

```
df_count.writeStream.queryName("group1_2").outputMode("complete").format("memory").start()
dfg = spark.sql("select * from group1_2")
dfg.coalesce(1).write.save("s3://results/group1_2", format='csv', header=True)
```

14. To connect S3, DynamoDB and CloudWatch to lambda function I also needed to create an IAM policy and a role to attach the policy to the Lambda function.
15. I also created a Cloudwatch to debug the system.

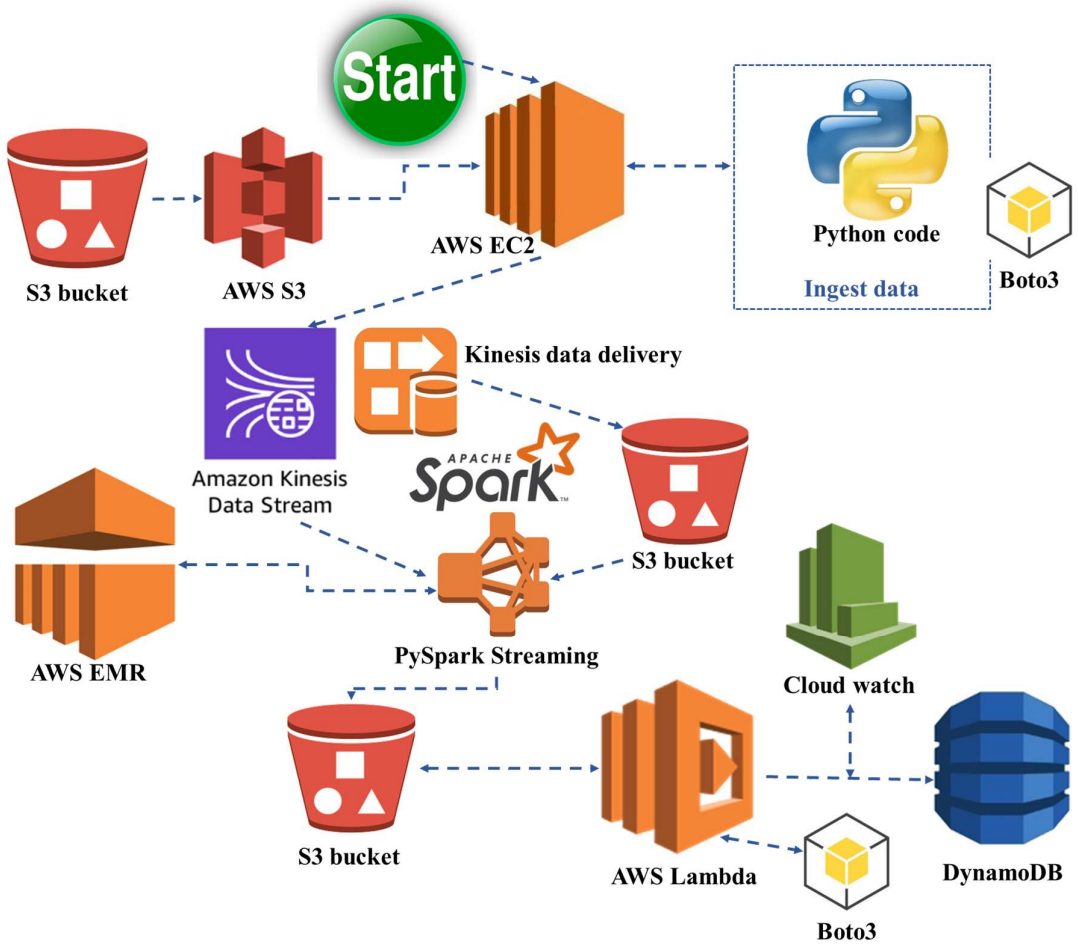

Fig. 3. System integration.

2. Approaches and algorithms used to answer each question.

Group1_Quesion2:

```
df_count= df.groupBy("UniqueCarrier").avg("ArrDelay").orderBy('avg(ArrDelay)', ascending=True)
df_count= df_count.writeStream.outputMode("complete").format("console").start()
df_count.awaitTermination()
```

To save the data into S3 and then lambda to DynamoDB:

```
df_count.writeStream.queryName("group1_2").outputMode("complete").format("memory").start()
dfg = spark.sql("select * from group1_2")
dfg.coalesce(1).write.save("s3://results/group1_2", format='csv', header=True)
```

Group1_Quesion3:

```
df_count = df.groupBy("DayOfWeek").avg("ArrDelay").orderBy('avg(ArrDelay)', ascending=True)
df_count.writeStream.outputMode("complete").format("console").start().awaitTermination()
```

To save the data into S3 and then lambda to DynamoDB:

```
df_count.writeStream.queryName("group1_3").outputMode("complete").format("memory").start()
dfg = spark.sql("select * from group1_3")
dfg.coalesce(1).write.save("s3://results/group1_3", format='csv', header=True)
```

Group2_Quesion1:

First:

```
df_count = df.groupBy("Origin", "UniqueCarrier").avg('DepDelay').orderBy(['Origin', 'avg(DepDelay)'], ascending=[True,True])
```

Then filter for example for SRQ

```
df_count = df_count.filter(df.Origin == "SRQ")
df_count.writeStream.outputMode("complete").format("console").start().awaitTermination()
```

To save the data into S3 and then lambda to DynamoDB:

```
df_count.writeStream.queryName("group2_1").outputMode("complete").format("memory").start()
dfg = spark.sql("select * from group2_1")
dfg.coalesce(1).write.save("s3://results/group2_1", format='csv', header=True)
```

Group2_Quesion2:

```
df_count = df.groupBy("Origin", "Dest").avg('DepDelay').orderBy(['Origin', 'avg(DepDelay)'], ascending=[True,True])
df_count = df_count.filter(df.Origin == "SRQ")
df_count.writeStream.outputMode("complete").format("console").start().awaitTermination()
```

To save the data into S3 and then lambda to DynamoDB:

```
df_count.writeStream.queryName("group2_2").outputMode("complete").format("memory").start()
dfg = spark.sql("select * from group2_2")
dfg.coalesce(1).write.save("s3://results/group2_2", format='csv', header=True)
```

Group2_Quesion3:

```
df3 = df.groupBy("Origin", "Dest", "UniqueCarrier").avg('ArrDelay').orderBy(['Origin', 'Dest', 'avg(ArrDelay)'], ascending=[True,True,True])
df_count = df3.filter(df3.Origin == "LGA").filter(df3.Dest == "BOS")
df_count.writeStream.outputMode("complete").format("console").start().awaitTermination()
```

To save the data into S3 and then lambda to DynamoDB:

```
df_count.writeStream.queryName("group2_3").outputMode("complete").format("memory").start()
dfg = spark.sql("select * from group2_3")
dfg.coalesce(1).write.save("s3://results/group2_3", format='csv', header=True)
```

Group2_Quesion4:

```
df_count = df.groupBy("Origin", "Dest").avg('ArrDelay').orderBy("Origin")
df_count = df3.filter(df3.Origin == "LGA").filter(df3.Dest == "BOS")
df_count.writeStream.outputMode("complete").format("console").start().awaitTermination()
```

To save the data into S3 and then lambda to DynamoDB:

```
df_count.writeStream.queryName("group2_4").outputMode("complete").format("memory").start()
dfg = spark.sql("select * from group2_4")
dfg.coalesce(1).write.save("s3://results/group2_4", format='csv', header=True)
```

Group3_Quesion2:

```
df2008 = df.filter(df.Year == "2008")

df_X = df2008.filter(df2008.DepTime < 1200.0)
df_Y = df2008.filter(df2008.DepTime > 1200.0)

oldColumns = df_X.schema.names
newColumns = ["ArrDelay_X","ArrTime_X", "Cancelled_X", "DayOfWeek_X", "DayofMonth_X", "DepDelay_X", "DepTime_X", "Dest_X", "Month_X", "Origin_X", "UniqueCarrier_X", "Year_X", "timeStam
df_X = reduce(lambda df_X, idx: df_X.withColumnRenamed(oldColumns[idx], newColumns[idx]), range(len(oldColumns)), df_X)

oldColumns = df_Y.schema.names
newColumns = ["ArrDelay_Y","ArrTime_Y", "Cancelled_Y", "DayOfWeek_Y", "DayofMonth_Y", "DepDelay_Y", "DepTime_Y", "Dest_Y", "Month_Y", "Origin_Y", "UniqueCarrier_Y", "Year_Y", "timeStam
df_Y = reduce(lambda df_Y, idx: df_Y.withColumnRenamed(oldColumns[idx], newColumns[idx]), range(len(oldColumns)), df_Y)

df4 = df_Y.alias('l').join(df_X.alias('r'), (f.col('l.Origin_Y') == f.col('r.Dest_X')) & (f.col('l.Month_Y') == f.col('r.Month_X')) & (f.col('l.DayofMonth_Y') == 2 + f.col('r.DayofMont
df4 = df4.withColumn('TotalArrivalDelay', df4.ArrDelay_X + df4.ArrDelay_Y)

df_count = df4.filter(df4.Origin_X == "BOS").filter(df4.Dest_X == "ATL").filter(df4.Dest_Y == "LAX").filter(df4.Month_X == 4).filter(df4.DayofMonth_X == 3)

df_count.writeStream.queryName("group3_2").outputMode("append").format("memory").start()
dfg = spark.sql("select * from group3_2")
dfg = dfg.sort(["TotalArrivalDelay"])
dfg.coalesce(1).write.save("s3://results/ group3_2", format='csv', header=True)
```

3. What are the results of each question? Use only the provided subset for questions from Group 2 and Question 3.2.

Group1_Quesion2:

UniqueCarrier	avg(ArrDelay)
HA	-1.01180434574519
AQ	1.1569234424812056
PS	1.4506385127822803
ML (1)	4.747609195734892
PA (1)	5.3224309999287875
F9	5.465881148819851
NW	5.557783392671835
WN	5.5607742598815735
OO	5.736312463662878
9E	5.8671846616957595
TZ	6.128612134539118
US	6.580217835501714
DH	6.798461072112793
EA	6.82503824598947
TW	6.8594851997016955
CO	6.959318619647949
AA	7.093206462455553
OH	7.349702615246028
DL	7.580375761024863
HP	7.725964027990898

Group1_Quesion3:

DayOfWeek	avg(ArrDelay)
6	4.301669926076596
2	5.990458841319885
7	6.613280292442754
1	6.716102802585582
3	7.203656394670348
4	9.094441008336657
5	9.721032337585571

Group2_Quesion1:

Origin	UniqueCarrier	avg(DepDelay)
SRQ	TZ	-0.38199697428139184
SRQ	XE	1.4897667777248929
SRQ	YV	3.4040219378427787
SRQ	AA	3.6334985133795836
SRQ	UA	3.9521220624342335
SRQ	US	3.9683982896741536
SRQ	TW	4.3046760650181035
SRQ	NW	4.856359241353663
SRQ	DL	4.8691794341573456
SRQ	MQ	5.350588235294118
SRQ	FL	5.938190895043385
SRQ	EA	7.055370012304447
SRQ	ML (1)	7.3514376996805115
SRQ	CO	8.133513782246036
SRQ	OH	9.708313310908217
SRQ	EV	17.74345076784101
SRQ	9E	20.28125
SRQ	B6	21.804093567251464

Origin	UniqueCarrier	avg(DepDelay)
CMH	DH	3.491114701130856
CMH	AA	3.5139262599083705
CMH	NW	4.0415550052579805
CMH	ML (1)	4.366459627329193
CMH	DL	4.713441339740944
CMH	PI	5.2012948793407885
CMH	EA	5.937389380530973
CMH	US	5.993296353520258
CMH	TW	6.159097425311084
CMH	YV	7.9611913357400725
CMH	OO	8.062805872756933
CMH	WN	8.130782646875877
CMH	9E	8.412538699690403
CMH	CO	8.49455665377453
CMH	UA	8.618599325742638
CMH	XE	9.202106422523578
CMH	OH	10.532717245491131
CMH	EV	11.371794871794872
CMH	HP	11.553977861819044
CMH	MQ	12.408363130711164

Origin	UniqueCarrier	avg(DepDelay)
JFK	UA	5.968325364871665
JFK	XE	8.113736263736264
JFK	CO	8.201208081649657
JFK	DH	8.742980908069951
JFK	AA	10.080735583893281
JFK	B6	11.127096222728753
JFK	PA (1)	11.523478655767484
JFK	NW	11.637817716512535
JFK	DL	11.986673184147481
JFK	TW	12.639071414087718
JFK	MQ	12.927430115417973
JFK	US	13.11160341080087
JFK	EA	14.005936675461742
JFK	EV	14.156153846153845
JFK	HP	14.177559971841664
JFK	PI	16.096052854480 77
JFK	OH	16.651307008911118
JFK	YV	18.080357142857142

Origin	UniqueCarrier	avg(DepDelay)
SEA	OO	2.7058196546578555
SEA	PS	4.720639332870048
SEA	YV	5.122262773722627
SEA	TZ	6.345003933910307
SEA	US	6.412384182257776
SEA	NW	6.498762407390315
SEA	DL	6.53562132870614
SEA	HA	6.855452674897119
SEA	AA	6.939152588687252
SEA	CO	7.096458868617853
SEA	F9	7.486242062728497
SEA	EV	7.896296296296296
SEA	WN	8.533977009882946
SEA	HP	8.993516068872793
SEA	TW	9.33205380270262
SEA	AS	9.508465269768392
SEA	UA	9.521271357608684
SEA	DH	10.841836734693878
SEA	PI	11.538356164383561
SEA	XE	11.821144098479364

Origin	UniqueCarrier	avg(DepDelay)
BOS	TZ	3.0637920850561136
BOS	PA (1)	4.447164795047816
BOS	ML (1)	5.734775641025641
BOS	EV	7.208137715179968
BOS	NW	7.245188786506978
BOS	DL	7.445339060304795
BOS	XE	8.10292249047014
BOS	US	8.687922116917663
BOS	AA	8.733506415381084
BOS	EA	8.891433950423595
BOS	B6	9.918861280563409
BOS	UA	10.066151559583444
BOS	TW	10.180054205741971
BOS	DH	10.181005763841151
BOS	FL	10.22269159741459
BOS	CO	10.3696471669923
BOS	9E	11.58736059479554
BOS	HP	11.632282612489965
BOS	PI	11.884455261640941
BOS	OH	12.594790937954045

Group2_Quesion2:

Origin	Dest	avg(DepDelay)
SRQ	EYW	0.0
SRQ	TPA	0.6741226378711916
SRQ	IAH	1.4606205250596658
SRQ	MEM	1.6875
SRQ	FLL	2.0
SRQ	BNA	2.068862275449102
SRQ	MCO	2.3138770331064715
SRQ	RDU	2.44759047440465
SRQ	MDW	2.777998674618953
SRQ	RSW	3.351498202157411
SRQ	CLT	3.3580551053484604
SRQ	CLE	3.3911310413552567
SRQ	BWI	3.7612801678908707
SRQ	MSP	4.028846153846154
SRQ	IND	4.474694589877836
SRQ	DCA	4.717703349282297
SRQ	DTW	4.859699279966116
SRQ	STL	5.182029598308668
SRQ	PIT	5.4664867296446245
SRQ	PBI	5.490854955338154

Origin	Dest	avg(DepDelay)
CMH	OMA	-5.0
CMH	AUS	-5.0
CMH	SYR	-5.0
CMH	CLE	1.047270214236351
CMH	SDF	1.3529411764705883
CMH	CAK	3.4973684210526317
CMH	SLC	3.9229390681003586
CMH	IND	4.140255009107468
CMH	DFW	4.149068101635722
CMH	MEM	4.157088812826518
CMH	IAD	4.157512953367876
CMH	DTW	4.325461833391426
CMH	MSP	4.339475642316278
CMH	CLT	4.401491935483871
CMH	BNA	4.418833296919379
CMH	IAH	4.5340599455040875
CMH	DAY	4.695861405197305
CMH	CVG	5.056123163051085
CMH	ATL	5.209974025974026
CMH	PIT	5.359655913978495

Origin	Dest	avg(DepDelay)
JFK	SWF	-10.5
JFK	MYR	0.0
JFK	ABQ	0.0
JFK	ANC	0.0
JFK	ISP	0.0
JFK	UCA	1.9079497907949792
JFK	BGR	3.210280373831776
JFK	BQN	3.6146839635661054
JFK	STT	4.33694150161216
JFK	CHS	4.339147286821706
JFK	AGS	4.75
JFK	SNA	4.905988023952096
JFK	PNS	5.65008576329331
JFK	SAV	5.830426356589148
JFK	STX	5.923423423423423
JFK	SRQ	6.2087821043910525
JFK	MLB	6.242894056847545
JFK	PSE	6.593548387096774
JFK	LGB	6.90639400921659
JFK	HPN	7.217391304347826

Origin	Dest	avg(DepDelay)
SEA	EUG	0.0
SEA	PSC	2.5949429846306398
SEA	CVG	3.7894085461177696
SEA	MEM	4.210309459104059
SEA	CLE	5.154081632653061
SEA	SNA	5.195116978439268
SEA	BLI	5.199197373221452
SEA	YKM	5.379647749510763
SEA	LIH	5.481081081081081
SEA	DTW	5.650894947511517
SEA	IAH	5.667182499208666
SEA	PIT	6.205256723716381
SEA	MKE	6.303830911492734
SEA	DFW	6.338736142223237
SEA	MSP	6.398255813953488
SEA	MCO	6.440322580645161
SEA	PHL	6.474397590361446
SEA	PDX	6.657576164273252
SEA	IAD	6.719712365171174
SEA	MDW	6.786296118830857

Origin	Dest	avg(DepDelay)
BOS	SWF	-5.0
BOS	ONT	-4.0
BOS	AUS	1.2166089965397924
BOS	LGA	3.0314992482928096
BOS	MSY	3.2839116719242902
BOS	LGB	4.986319389634307
BOS	OAK	5.393518518518518
BOS	MDW	5.837130801687763
BOS	DCA	5.8880312665433605
BOS	BDL	5.957978421351505
BOS	MKE	5.974571078431373
BOS	RSW	6.678701579160481
BOS	SJC	6.751966193215166
BOS	PDX	7.01158940397351
BOS	MYR	7.275862068965517
BOS	SJU	7.307225567392312
BOS	MSP	7.339358439338327
BOS	DTW	7.494952728916344
BOS	CLE	7.543952609983425
BOS	DFW	7.560083676537883

Group2_Quesion3:

Origin	Dest	UniqueCarrier	avg(ArrDelay)
LGA	BOS	TW	-3.0
LGA	BOS	US	-2.9042429927168394
LGA	BOS	PA (1)	-0.4187248422203449
LGA	BOS	DL	1.747324224799532
LGA	BOS	EA	4.8213728549141965
LGA	BOS	MQ	9.866226864577607
LGA	BOS	NW	14.444444444444445
LGA	BOS	OH	27.984848484848484
LGA	BOS	AA	28.5

Origin	Dest	UniqueCarrier	avg(ArrDelay)
BOS	LGA	TW	-11.0
BOS	LGA	US	1.0952367939223349
BOS	LGA	DL	2.0246025602857993
BOS	LGA	PA (1)	6.071749550629119
BOS	LGA	EA	9.480668069437138
BOS	LGA	MQ	12.643273798785255
BOS	LGA	NW	15.22985468956407
BOS	LGA	AA	28.0
BOS	LGA	OH	30.448275862068964
BOS	LGA	TZ	133.0

Origin	Dest	UniqueCarrier	avg(ArrDelay)
OKC	DFW	TW	0.10124333925399645
OKC	DFW	EV	1.358974358974359
OKC	DFW	AA	4.570106940850489
OKC	DFW	MQ	4.675752473163545
OKC	DFW	DL	6.7315385583092215
OKC	DFW	OO	12.835087719298246
OKC	DFW	OH	47.5

Origin	Dest	UniqueCarrier	avg(ArrDelay)
MSP	ATL	EA	4.2015625
MSP	ATL	OO	4.766
MSP	ATL	FL	6.292677547419497
MSP	ATL	DL	6.34326262780406
MSP	ATL	NW	7.015818604943707
MSP	ATL	OH	8.303473491773309
MSP	ATL	EV	10.12092731829574

Group2_Quesion4:

Origin	Dest	avg(ArrDelay)
LGA	BOS	1.4838648387077622

Origin	Dest	avg(ArrDelay)
BOS	LGA	3.7841181478417854

Origin	Dest	avg(ArrDelay)
OKC	DFW	4.969055284955558

Origin	Dest	avg(ArrDelay)
MSP	ATL	6.737007973674219

Group3_Quesion2:

BOS → ATL → LAX, 03/04/2008:

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
3	4	2008	548.0	BOS	ATL	7.0	5	4	2008	1857.0	ATL	LAX	-2.0	5.0
3	4	2008	548.0	BOS	ATL	7.0	5	4	2008	1704.0	ATL	LAX	0.0	7.0
3	4	2008	853.0	BOS	ATL	13.0	5	4	2008	1857.0	ATL	LAX	-2.0	11.0
3	4	2008	548.0	BOS	ATL	7.0	5	4	2008	1451.0	ATL	LAX	5.0	12.0
3	4	2008	548.0	BOS	ATL	7.0	5	4	2008	1332.0	ATL	LAX	6.0	13.0
3	4	2008	853.0	BOS	ATL	13.0	5	4	2008	1704.0	ATL	LAX	0.0	13.0
3	4	2008	713.0	BOS	ATL	17.0	5	4	2008	1857.0	ATL	LAX	-2.0	15.0
3	4	2008	548.0	BOS	ATL	7.0	5	4	2008	1943.0	ATL	LAX	9.0	16.0
3	4	2008	713.0	BOS	ATL	17.0	5	4	2008	1704.0	ATL	LAX	0.0	17.0
3	4	2008	853.0	BOS	ATL	13.0	5	4	2008	1451.0	ATL	LAX	5.0	18.0
3	4	2008	853.0	BOS	ATL	13.0	5	4	2008	1332.0	ATL	LAX	6.0	19.0
3	4	2008	548.0	BOS	ATL	7.0	5	4	2008	2134.0	ATL	LAX	12.0	19.0
3	4	2008	556.0	BOS	ATL	23.0	5	4	2008	1857.0	ATL	LAX	-2.0	21.0
3	4	2008	713.0	BOS	ATL	17.0	5	4	2008	1451.0	ATL	LAX	5.0	22.0
3	4	2008	853.0	BOS	ATL	13.0	5	4	2008	1943.0	ATL	LAX	9.0	22.0
3	4	2008	556.0	BOS	ATL	23.0	5	4	2008	1704.0	ATL	LAX	0.0	23.0
3	4	2008	713.0	BOS	ATL	17.0	5	4	2008	1332.0	ATL	LAX	6.0	23.0
3	4	2008	853.0	BOS	ATL	13.0	5	4	2008	2134.0	ATL	LAX	12.0	25.0
3	4	2008	834.0	BOS	ATL	28.0	5	4	2008	1857.0	ATL	LAX	-2.0	26.0
3	4	2008	713.0	BOS	ATL	17.0	5	4	2008	1943.0	ATL	LAX	9.0	26.0

PHX → JFK → MSP, 07/09/2008:

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
7	9	2008	1127.0	PHX	JFK	-25.0	9	9	2008	1747.0	JFK	MSP	-17.0	-42.0
7	9	2008	634.0	PHX	JFK	-19.0	9	9	2008	1747.0	JFK	MSP	-17.0	-36.0
7	9	2008	904.0	PHX	JFK	43.0	9	9	2008	1747.0	JFK	MSP	-17.0	26.0
7	9	2008	1127.0	PHX	JFK	-25.0	9	9	2008	1239.0	JFK	MSP	139.0	114.0
7	9	2008	634.0	PHX	JFK	-19.0	9	9	2008	1239.0	JFK	MSP	139.0	120.0
7	9	2008	904.0	PHX	JFK	43.0	9	9	2008	1239.0	JFK	MSP	139.0	182.0

DFW → STL → ORD, 24/01/2008:

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
24	1	2008	657.0	DFW	STL	-14.0	26	1	2008	1654.0	STL	ORD	-5.0	-19.0
24	1	2008	836.0	DFW	STL	-9.0	26	1	2008	1654.0	STL	ORD	-5.0	-14.0
24	1	2008	657.0	DFW	STL	-14.0	26	1	2008	1330.0	STL	ORD	0.0	-14.0
24	1	2008	940.0	DFW	STL	-8.0	26	1	2008	1654.0	STL	ORD	-5.0	-13.0
24	1	2008	836.0	DFW	STL	-9.0	26	1	2008	1330.0	STL	ORD	0.0	-9.0
24	1	2008	940.0	DFW	STL	-8.0	26	1	2008	1330.0	STL	ORD	0.0	-8.0
24	1	2008	657.0	DFW	STL	-14.0	26	1	2008	1452.0	STL	ORD	79.0	65.0
24	1	2008	836.0	DFW	STL	-9.0	26	1	2008	1452.0	STL	ORD	79.0	70.0
24	1	2008	940.0	DFW	STL	-8.0	26	1	2008	1452.0	STL	ORD	79.0	71.0

LAX → MIA → LAX, 16/05/2008:

DayofMonth_X	Month_X	Year_X	DepTime_X	Origin_X	Dest_X	ArrDelay_X	DayofMonth_Y	Month_Y	Year_Y	DepTime_Y	Origin_Y	Dest_Y	ArrDelay_Y	TotalArrivalDelay
16	5	2008	817.0	LAX	MIA	10.0	18	5	2008	1925.0	MIA	LAX	-19.0	-9.0
16	5	2008	817.0	LAX	MIA	10.0	18	5	2008	2101.0	MIA	LAX	-13.0	-3.0
16	5	2008	556.0	LAX	MIA	19.0	18	5	2008	1925.0	MIA	LAX	-19.0	0.0
16	5	2008	817.0	LAX	MIA	10.0	18	5	2008	1536.0	MIA	LAX	-5.0	5.0
16	5	2008	556.0	LAX	MIA	19.0	18	5	2008	2101.0	MIA	LAX	-13.0	6.0
16	5	2008	817.0	LAX	MIA	10.0	18	5	2008	1841.0	MIA	LAX	2.0	12.0
16	5	2008	556.0	LAX	MIA	19.0	18	5	2008	1536.0	MIA	LAX	-5.0	14.0
16	5	2008	708.0	LAX	MIA	34.0	18	5	2008	1925.0	MIA	LAX	-19.0	15.0
16	5	2008	708.0	LAX	MIA	34.0	18	5	2008	2101.0	MIA	LAX	-13.0	21.0
16	5	2008	556.0	LAX	MIA	19.0	18	5	2008	1841.0	MIA	LAX	2.0	21.0
16	5	2008	708.0	LAX	MIA	34.0	18	5	2008	1536.0	MIA	LAX	-5.0	29.0
16	5	2008	708.0	LAX	MIA	34.0	18	5	2008	1841.0	MIA	LAX	2.0	36.0
16	5	2008	817.0	LAX	MIA	10.0	18	5	2008	1651.0	MIA	LAX	41.0	51.0
16	5	2008	556.0	LAX	MIA	19.0	18	5	2008	1651.0	MIA	LAX	41.0	60.0
16	5	2008	708.0	LAX	MIA	34.0	18	5	2008	1651.0	MIA	LAX	41.0	75.0

4. What system- or application-level optimizations (if any) did you employ?

- I used Kinesis which I found it more reliable and faster than Kafka.
- To make the queries faster I employed the AWS-EMR (elastic map reduce) which is kind of parallelism and then using PySpark Streaming on it.
- To obtain the results using PySpark Streaming I used the In-Memory Computation capability of Spark which makes the computation faster and used in memory data to inquiry.
- I used automatic capability of AWS-Lambda function. I created a Lambda function and set its trigger on AWS-S3 buckets as whenever I store the data on a specific folder in AWS-S3 bucket my lambda function automatically read the data and push it into the DynamoDB tables.

5. Give your opinion about whether the results make sense and are useful in any way.

Yes it is interesting to me. I did some researches for example according to the ClaimCompass the list of the top 10 most popular airports in the USA in 2020 was as the following:

ATL, LAX, ORD, DFW, DEN, JFK, SFO, LAS, SEA, CLT

You can compare it with our results for 2008:

ORD, ATL, DFW, LAX, PHX, DEN, DTW, IAH, MSP, SFO

Also, according to the AFAR data Hawaiian Airlines: (87.4%) is the highest scored U.S.-based carriers in OAG's most recent report. In our data also (Group 1 question 2) HA: -1.01 was the top airline by on-time arrival performance. In addition, I think especially Group 3 question 2 can be very useful to find the possible route having two destination and specific time frame.

6. **How did the different stacks (e,.g., Hadoop and Spark Streaming, or any combination of your choice) from Task 1 and Task 2 compare? Which stack did you find the easiest to use? The fastest?**

I used EMR and PySpark for Task 1 and Task 2. The difference was for Task 1 I used batch processing which I had the complete data in a single csv file and I used PySpark batch processing. In Task 2 however I ingest the data using kinesis and then received the kinesis streamed data in PySpark Streaming. I think we cannot compare the two tasks as the nature is different. Task 2 is a real time processing so it is more challenging but practically there are many cases that we have live data and we want to analysis them as real-time. I had lots of challenges for Task 2 to receive and analysis the data in compare to Task 1.