Nathaniel Rupsis

CS-598 Cloud Computing Capstone

Task 1

Video Link: https://mediaspace.illinois.edu/media/t/1_3e7cb3i0

Data Extraction and Cleaning

Given the data is stored as an EBS volume, the only way I found to access the data, was to spin up an EC2 instance and attach the EBS volume. Once the volume was attached to the EC2 server, I mounted the volume, zipped up the data, and downloaded it locally using SCP

```
# zipping up the airline_ontime data
> zip -r data.zip .

# downloading the data locally
> scp -i 'key.pem'  user@ec2-xx-xx-xxx-xxx.compute-1.amazonaws.com:/path/to/file ./local/path
```

Once the data was locally on my machine, I used some simple bash commands to aggregate the data by year, and create a sample sub set to explore / work with

---

```
#grab a random subset of the data to observe. Ensure the header information is preserved
cat 1988.csv | head -n 1 > 1988_1_sample.csv
shuf -n 50000 On_Time_On_Time_Performance_1988_1.csv >> 1988_1_sample.csv

# grabs all of the lines minus the first, and sticks it into a file.
# Used to interate over each year dir and aggregate into a single year file.
awk -v ORS="" -F "\\"*,\\"*" 'NR>1  {print $10}' file.csv
```

Once I had aggregated data, I attempted to load the data into OpenRefine (I just so happened to be taking CS-513 at the same time as this course). Unfortunately, even the sample set I had created (around 100mb) was causing issues due to data size.

My second attempt at cleaning data was to explore a cloud based approach using AWS Glue and their ETL job framework. After loading the data into S3, and crawling the data to create tables, I spun up a development endpoint to explore their note book offerings. I made the unfortunately mistake of forgetting about the dev endpoint. After 5 days of up time, I had racked up a $250 AWS bill exclusively on the AWS Glue Development endpoint. A word of warning, **always set up an AWS billing budget with alerts.** After realizing my mistake, I decided to process the data locally with a python script to avoid any more costly mistakes.

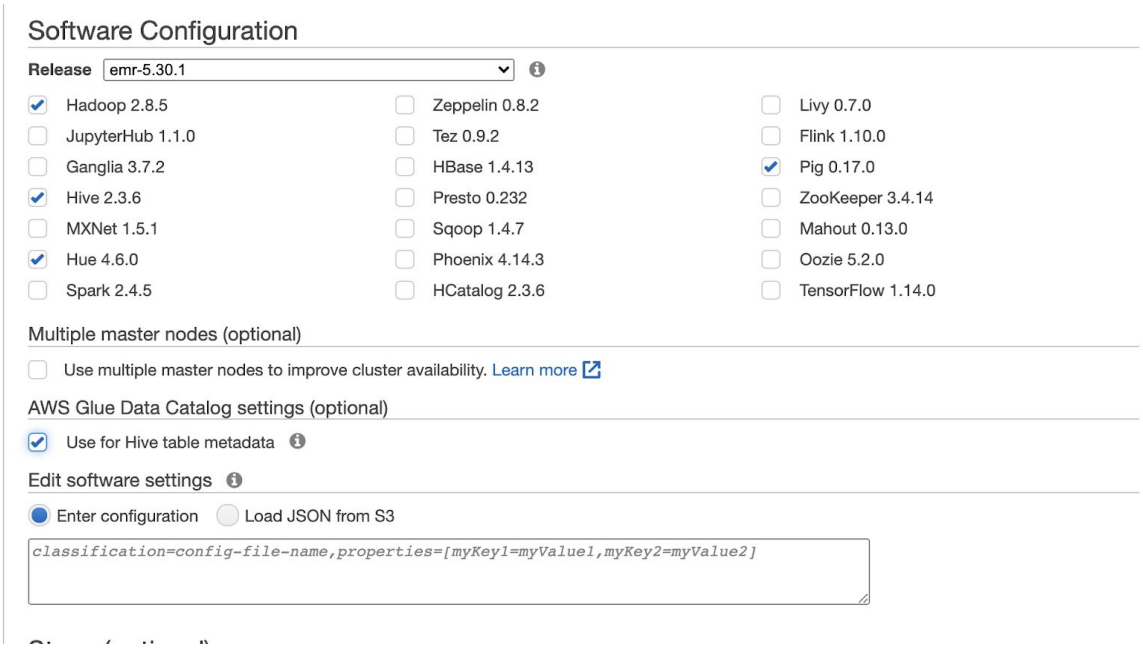
The depth of cleaning was pretty minimal. For each csv file, I encoded to utf-8, stripped any leading / trailing white space (and special characters) and removed field level commas. I.e, Chicago, IL became Chicago IL

With the cleansed data, I uploaded to a new S3 bucket, crawled the bucket with AWS Glue to create a schema, and ensured the date was injected correctly using some sample queries via AWS Athena.

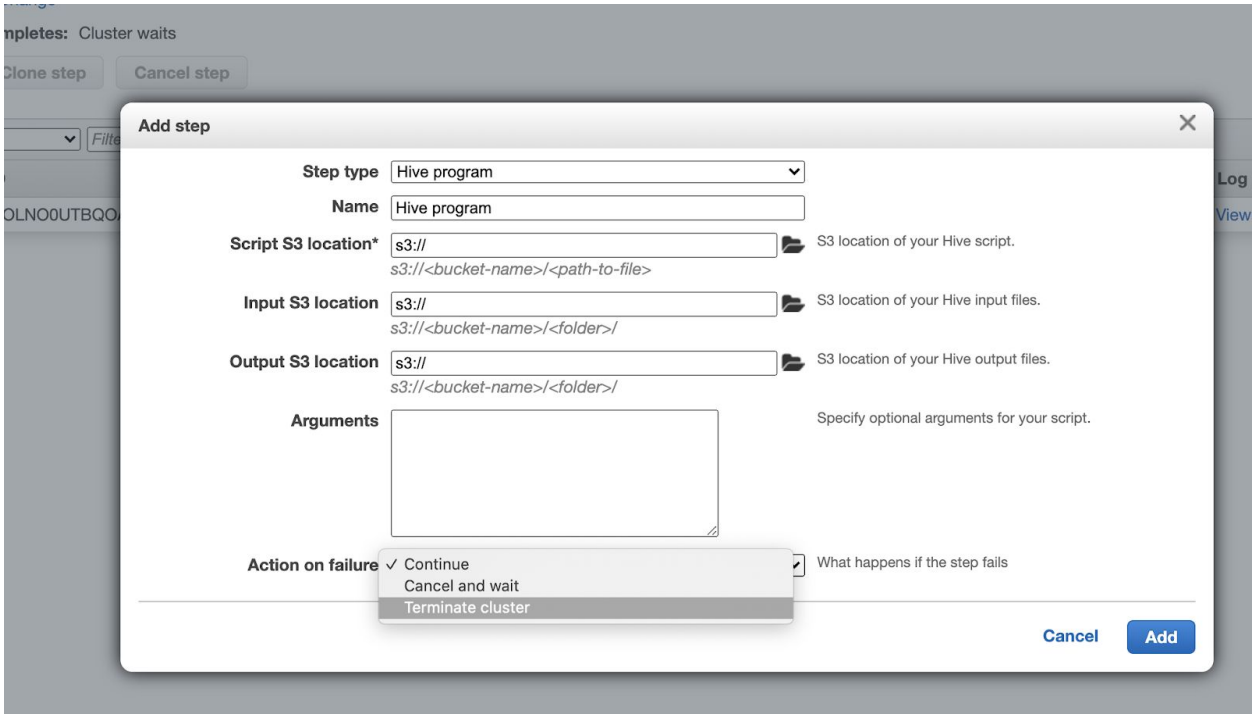
System Integration

For task 1, my batch processing platform of choice is apache Hive. Hive allows me the flexibility of Hadoop map reduce jobs with the ease and convenience of familiar SQL queries.

AWS offers some incredibly powerful data analytics tool for data processing. One such product is called EMR (Elastic Map Reduce), which provides managed Hadoop Clusters running on EC2 instances. EMR allows for the option to utilize AWS Glue settings, and since I crawled my data and created a schema, creating the apache Hive meta store was a breeze



Once I have a hadoop cluster up and running, it's just a matter of SSH'ing into the master node, spinning up hive, and executing my queries. For longer running job, there is an option to create "Steps" which are just queued hadoop task, with the option to terminate the cluster on failure / competition



Hive allows you to create external tables, which allows us to save the data in either S3 or dynamoDB.

Solution Approach

For each question in task 1, it's as simple as writing an SQL query to get the results.
There are some slight performance issues with the system, mapping an internal Hive table to an external DynamoDB causes the biggest bottleneck. Underneath the hood, the dynamoDB driver is making API calls to the dynamoDB system. The only way i found to improve this bottleneck was to increase the write units per table, and increase the number of available map / reduce partitions to allow for more parallel writes .

Question 2.1

For question 2.1 & 2.2, I've created a reference table in dynamoDB containing the airport codes:

CREATE EXTERNAL TABLE ddb_cs598_question_2_1_2

```
   (airport_code STRING)
STORED BY 'org.apache.hadoop.hive.dynamodb.DynamoDBStorageHandler'
TBLPROPERTIES(
   "dynamodb.table.name" = "q_2_1-2_ref",
   "dynamodb.column.mapping"="airport_code:airport_code"
);
```

Query: (rerun the query increasing the offset each run)

```
INSERT OVERWRITE TABLE ddb_cs598_question_2_1  select origin, carrier, avg(depdelay) as avg_depdelay from cs598.cs598task1 where origin IN (select * from ddb_cs598_question_2_1_2 limit 1) group by origin, carrier having avg_depdelay is not null sort by avg_depdelay asc limit 10;
```

Results:

Airport Code	Carrier	Average Departure Delay
CMI	1) OH 2) US 3) TW 4) PI 5) DH 6) EV 7) MQ	1) 0.6116264687693259 2) 2.033047346679377 3) 4.735353535353536 4) 5.587096774193548 5) 6.027888446215139 6) 6.665137614678899 7) 8.016004886988393
BWI	F9 PA CO YV NW AA 9E US UA FL	1) 0.7562437562437563 2) 4.761904761904762 3) 5.1485301783373085 4) 5.496503496503497 5) 5.791685678899681 6) 6.103507291505722 7) 7.239805825242718 8) 7.508907895286403 9) 7.734484766155638 10) 7.747100147034798
MIA	1) 9E 2) EV 3) RU 4) TZ 5) XE 6) PA 7) NW 8) US 9) UA 10) ML	1) -3.0 2) 1.2026431718061674 3) 1.3021664766248575 4) 1.782243551289742 5) 2.745644599303136 6) 3.7389675499972452 7) 4.5040642076502735 8) 6.061599120445843 9) 6.88091441401217 10) 7.504550050556118

LAX	1) RU 2) MQ 3) OO 4) FL 5) TZ 6) NW 7) F9 8) HA 9) YV 10) EA	1) 1.9483870967741936 2) 2.407221858260434 3) 4.2219592877139975 4) 4.725127379994636 5) 4.763940985246312 6) 4.951674289186682 7) 5.729155372438469 8) 5.813645621181263 9) 6.024156085475379 10) 6.4884189325276935
IAH	1) NW 2) PA 3) RU 4) US 5) F9 6) PI 7) AA 8) WN 9) TW 10) OO	1) 3.51326380228999 2) 3.9847272727272727 3) 4.798695924258635 4) 5.042346298619824 5) 5.545243619489559 6) 5.745321399511798 7) 5.785937409374508 8) 6.449623532414497 9) 6.457104557640751 10) 6.58795822240426
SFO	TZ MQ F9 PA NW DL CO US AA TW	1) 3.952415634862831 2) 4.853923777799549 3) 5.162444663059518 4) 5.305942773294204 5) 5.590084712688538 6) 6.570832797840895 7) 6.837172913980928 8) 7.137510103662098 9) 8.02075832871468 10) 8.069147860259486

Question 2.2 (rerun the query increasing the offset each run)

Query:

```
INSERT OVERWRITE TABLE ddb_cs598_question_2_2
select origin, dest, avg(depdelay) as avg_depdelay
from cs598.cs598task1 where origin  IN (select * from ddb_cs598_question_2_1_2 limit 1)
group by origin, dest having avg_depdelay is not null
sort by avg_depdelay asc limit 10;
```

Results:

Airport Code	Destination	Average Departure Delay
CMI	1) ABI 2) PIT 3) PIA 4) CVG	1) -7.0 2) 1.1024305555555556 3) 1.4523809523809523 4) 1.8947616800377536

	5) DAY 6) STL 7) DFW 8) ATL 9) ORD	5) 2.89157004073958 6) 5.039735099337748 7) 5.944142746314973 8) 6.665137614678899 9) 8.194098143236074
BWI	1) SAV 2) MLB 3) DAB 4) SRQ 5) IAD 6) UCA 7) GSP 8) BGM 9) SJU 10) OAJ	1) -7.0 2) 1.155367231638418 3) 1.4695945945945945 4) 1.5884838880084522 5) 2.262295081967213 6) 3.6748726655348047 7) 4.197686645636172 8) 4.3842260649514975 9) 4.409805634417995 10) 4.453125
MIA	1) SHV 2) BUF 3) SAN 4) SLC 5) HOU 6) ISP 7) MEM 8) PSE 9) TLH 10) MCI	1) 0.0 2) 1.0 3) 1.710382513661202 4) 2.5371900826446283 5) 2.8840482573726542 6) 3.647398843930636 7) 3.7935208981468955 8) 3.975845410628019 9) 4.389016018306636 10) 5.5
LAX	1) SDF 2) IDA 3) DRO 4) RSW 5) LAX 6) BZN 7) MAF 8) PIH 9) IYK 10) MFE	1) -16.0 2) -7.0 3) -6.0 4) -3.0 5) -2.0 6) -0.7272727272727273 7) 0.0 8) 0.0 9) 1.2698247440569148 10) 1.3764705882352941
IAH	1) MSN 2) AGS 3) MLI 4) EFD 5) JAC 6) HOU 7) MTJ 8) RNO 9) BPT 10) VCT	1) -2.0 2) -0.6187904967602592 3) -0.5 4) 1.8877082136703045 5) 2.570588235294118 6) 2.631741821396994 7) 2.9501569858712715 8) 3.22158438576349 9) 3.5995325282430852 10) 3.6119087837837838
SFO	1) SDF 2) MSO 3) PIH 4) OAK 5) LGA 6) PIE	1) -10.0 2) -4.0 3) -3.0 4) -2.6271820448877805 5) -1.7575757575757576

	7) FAR 8) BNA 9) MEM 10) SJC	6) -1.3410404624277457 7) 0.0 8) 2.425966447848286 9) 3.165533496509836 10) 4.089209855564996

Question 2.3

For Question 2.3 (and 2.4) create a reference table in dynamoDB

Query: (for each X,Y airports listed in 2.3 / 2.4 task):

select origin, dest, carrier, avg(arrdelay) as avg_arrdelay
from cs598.cs598task1 where origin == 'IND' and Dest == 'CMH' group by
origin, dest, carrier having avg_arrdelay is not null
sort by avg_arrdelay asc limit 10;

Results:

To / From	Carrier	Average Arrival Delay
CMI -> ORD	1) MQ	1) 10.14366290643663
IND -> CMH	1) CO 2) NW 3) AA 4) HP 5) US 6) DL	1) -2.6579673776662482 2) 3.941798941798942 3) 5.5 4) 5.697254901960784 5) 6.237020316027088 6) 10.6875
DFW -> IAH	1) PA 2) EV 3) CO 4) OO 5) RU 6) XE 7) AA 8) DL 9) MQ	1) -1.5964912280701755 2) 5.0925133689839575 3) 6.543978685024906 4) 7.564007421150278 5) 7.7914915966386555 6) 8.442865779927448 7) 8.545200789587318 8) 8.904884655714785 9) 9.103211009174313
LAX -> SFO	1) TZ 2) F9 3) EV 4) MQ 5) AA 6) WN 7) US 8) CO	1) -7.619047619047619 2) -2.028685790527018 3) 6.964630225080386 4) 7.8077634011090575 5) 8.181292220175184 6) 8.79205149734117 7) 8.822730864755023

	9) UA 10) PA	8) 9.659957627118644 9) 10.364480979470095 10) 10.379324462640737
JFK -> LAX	1) UA 2) HP 3) AA 4) DL 5) PA 6) TW	1) 3.329564447940222 2) 6.680599369085174 3) 7.214716627006736 4) 7.934460351304701 5) 9.1992700729927 6) 12.125
ATL -> PHX	1) FL 2) US 3) HP 4) EA 5) DL	1) 4.552631578947368 2) 6.28811524609844 3) 8.481436314363144 4) 8.662650602409638 5) 9.900493798805696

Question 3.2

Query:

```
select *, (a.ArrDelay + b.ArrDelay) as total_delay  from
   (select origin, dest, Carrier, FlightNum, ArrDelay, CRSDepTime, flightdate
   from cs598.cs598task1 where
   origin = <origin> and
   Dest = <lay over> and
   Year = <departure year> and
   Month = <departure month> and
   DayofMonth = <departure day> and
   CRSDepTime < 1200 and
   ArrDelay is not null
   sort by ArrDelay asc limit 1) as a
JOIN
   (select origin, dest, Carrier, FlightNum, ArrDelay, CRSDepTime, flightdate
      from cs598.cs598task1 where
      origin = <layover> and
      Dest = <dest >and
      Year = <departure year> and
      Month = <departure month> and
      DayofMonth = <departure day + 2> and
      CRSDepTime > 1200 and
      ArrDelay is not null
      sort by ArrDelay asc limit 1) as b
 ON (a.dest = b.origin);
```

Results:

Leg 1	Leg 2
CMI → ORD Carrier: MQ Flight Number: 4278 Sched Depart: 2008-03-04 7:10 Arrival Delay: -14.0	ORD -> LAX: Carrier: AA Flight Number: 607 Sched Depart: 2008-03-06 19:50 Arrival Delay: -24.0 Total arrival delay: -38.0
JAX → DFW Carrier: AA Flight Number: 845 Sched Depart: 2008-09-09 7:25 Arrival Delay: 1.0	DFW -> CRP: Carrier: MQ Flight Number: 3627 Sched Depart: 2008-09-11 16:45 Arrival Delay: -7.0 Total arrival delay: -6.0
SLC → BFL Carrier: OO Flight Number: 3755 Sched Depart: 2008-04-01 11:00 Arrival Delay: 12.0	BFL -> LAX: Carrier: OO Flight Number: 5429 Sched Depart: 2008-04-03 14:55 Arrival Delay: 6.0 Total arrival delay: 18.0
LAX → SFO Carrier: WN Flight Number: 3534 Sched Depart: 2008-07-12 6:50 Arrival Delay: -13.0 WRONG	SFO -> PHX: Carrier: US Flight Number: 412 Sched Depart: 2008-07-15 19:25 Arrival Delay: -19.0 Total arrival delay: -32.0
DFW → ORD Carrier: UA Flight Number: 1104 Sched Depart: 2008-06-10 7:00 Arrival Delay: -21.0	ORD -> DFW: Carrier: AA Flight Number: 2341 Sched Depart: 2008-06-12 16:45 Arrival Delay: -10.0

	Total arrival delay: -31.0
LAX → ORD Carrier: UA Flight Number: 944 Sched Depart: 2008-01-01 7:05 Arrival Delay: 1.0	ORD -> JFK: Carrier: B6 Flight Number: 918 Sched Depart: 2008-01-03 19:00 Arrival Delay: -7.0 Total arrival delay: -6.0

Conclusion

In summary, the answers obtained using my solution fall within the 10% threshold. The major discrepancies from the sample solutions are with question 1.1, 1.2, and 1.3. My assumption is that there is some data rot that I failed to clean up, resulting in a different aggregate count.

Overall the system is incredibly flexible. AWS makes integration between various services super easy and intuitive. If I were to implement this pipeline in production, I'd set up a daily crawler that would trigger a Lambda function to run a EMR batch job that would feed the results into whatever data store was needed.