**ETL PROCESS DOCUMENTATION**

**TASK 1. Selecting data elements from alaskaair.com & Modeling these events as a JSON Schema.**

In this step, I will extract the data from source system into the staging area. Data cleaning & transformations will be done directly in staging area. The redundant events will be removed from the staging area only. Then we validate extracted data before it moves into the Data Warehouse.

* Input Schema – Data collected from flight search input
* Output Schema – Data collected from flight search results

**Data Model – JSON Schema for search input**

{

  "$id": "The unique search ID",

  "customer\_id": "The unique customer id for each customer",

  "properties": {

    "from": {

      "type": "string",

      "description": "3 characters of city name or code for source"

    },

    "to": {

      "type": "string",

      "description": "3 characters of city name or code for destination"

    },

    "flight\_type”: {

      "type": "string",

      "description": "Round-trip, One-way or Multi-city"

    },

    "departure\_date": {

      "type": "date",

      "description": "The date of departure"

    },

    "num\_adults": {

      "type": "int",

      "description": "Number of adults"

    },

    "num\_children": {

      "type": "int",

      "description": "Number of children"

    }

  }

}

**Data Model – JSON schema for search output**

{

  "$id": "The unique search ID",

  "properties": {

    "eta": {

      "type": "time",

      "description": "The estimated time arrival"

    },

    "stops": {

      "type": "int",

      "description": "The number of stops, can be 0 1, 2 or so"

    },

    "main\_price": {

      "type": "float",

      "description": "Main price"

    },

    "first\_class\_price": {

      "type": "float",

      "description": "First class price"

    },

    "is\_refundable": {

      "type": "binary",

      "description": "Is refundable or not"

    },

    "stop\_times": {

      "type": "int",

      "description": "The average time between stops"

    },

    "is\_converted": {

      "type": "binary",

      "description": "Whether the listing got booked or not"

    }},

    "is\_available": {

      "type": "binary",

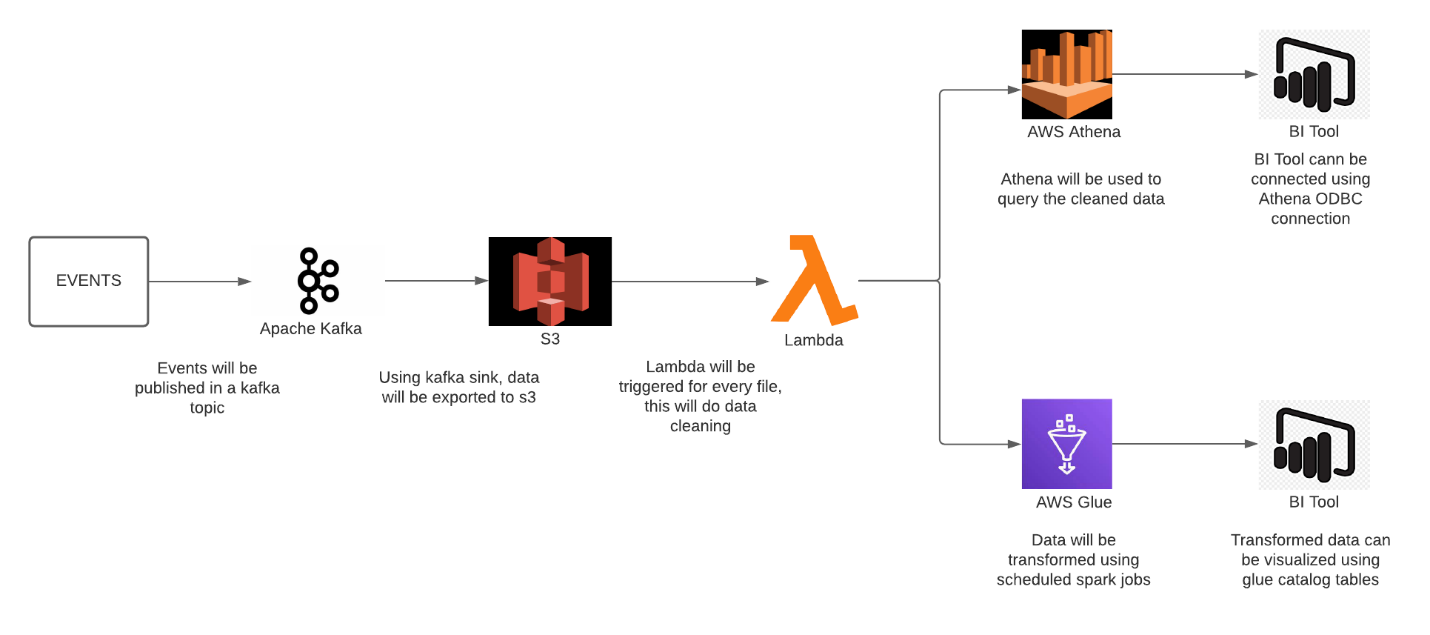
      "description": "Whether the seats are available or not"

    }

  }

}

**TASK 2. Describe the process and tools to model data**



**TASK 3. Real-time Data Consumption**

We can build live dashboards and real-time applications on data stored in DynamoDB & Redshift to handle real-time data

**Approach 1:**

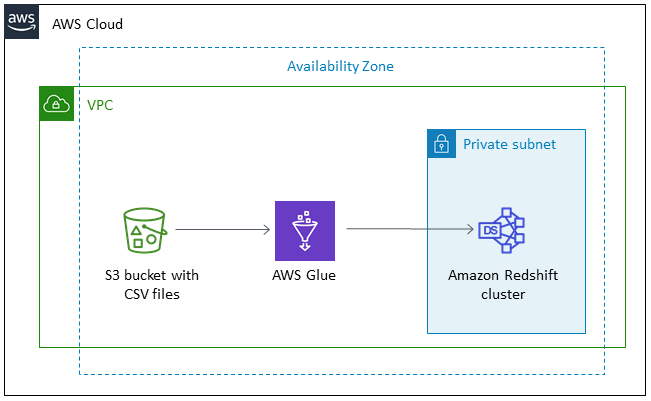
**S3 🡪 Lambda 🡪 DynamoDB**



In this scenario, changes to our S3 bucket will trigger a call to a Lambda function, which will take those changes and update a separate aggregate table stored in Dynamo DB. The Lambda will use the Dynamo DB Streams API to efficiently iterate through the recent changes to the table without having to do a complete scan. Dynamo DB can be directly connected to various BI tools for Real-time consumption.

**Approach 2:**

**S3 🡪 Glue 🡪 Redshift**



When moving data to and from an Amazon Redshift cluster, AWS Glue jobs issue COPY and UNLOAD statements against Amazon Redshift to achieve maximum throughput. These commands require that the Amazon Redshift cluster access S3 as a staging directory. The advantage of AWS Glue vs. setting up your own AWS data pipeline, is that Glue automatically discovers data model and schema, and even auto-generates ETL scripts.

**TASK 4. Structure Walkthrough with Pros & Cons**

Streaming data processing requires two layers: a storage layer and a processing layer. The storage layer needs to support record ordering and strong consistency. The processing layer is responsible for consuming data from the storage layer and running computations on that data.

* Consuming Streaming Data:
  + When records are generated using events, they get transferred in a Kafka topic which are buffered in their respective partitions for consumption. The application then deliver the records to a data lake (S3).
* Server less stream processing with AWS Lambda:
  + AWS Lambda continuously polls every PUT record of your stream and invokes the pre-processing function that will clean the data and re-write it in a different folder.
* Time-Based Scheduled processing using AWS Glue:
  + AWS Glue tracks data that has already been processed during a previous run of an ETL job by persisting state information from the job run. This persisted state information i.e. job bookmarks help AWS Glue maintain state information and prevent the reprocessing of old data. ETL job will take the data from S3 which is already cleaned using Lambda function and it will prevent duplicate processing and duplicate data in the job's target data store.

**Pros of Architecture:**

* + - This is a cost-efficient framework, as we’re using either open source technologies or server less AWS services.
    - Easy to setup

**Cons of Architecture:**

* + - You also have to plan for scalability, data durability, and fault tolerance in both the storage and processing layers.

**TASK 5. Exercise - Recommendation on travel answers**

1. **What is the conversion rate for an individual flight search result? For definition, an ‘individual flight search result’ is the following:**

Athena Query Logic:

- Getting all the records for unique searches – VAR 1

Select count(\*) from search\_input

- Getting converted records for all searches – VAR 2

SELECT count(\*) FROM (SELECT DISTINCT \_id from search\_output WHERE is\_converted=True)

- Dividing VAR 2 by VAR 1 will give me the **conversion rate.**

1. **How does the cost in miles affect the guests’ willingness to purchase? For this question, assume that the conversion space is already tagged successfully, and this data is already accessible**

Athena Query Logic:

- Getting the prices in bins and grouping it by conversion. This can be done in glue job using custom python function

groups = df.groupby(['is\_converted', pd.cut(df.cost, bins)])

groups.size().unstack()

This will give a table like this:

|  |  |  |
| --- | --- | --- |
| price | is\_converted | |
| 100-200 | yes | no |
| 200-300 | 10% | 90% |
| 300-400 | 9% | 91% |
| 400-500 | 8% | 92% |
| 500-600 | 7% | 93% |
| 600-700 | 6% | 94% |
| 700-800 | 5% | 95% |

Using the above table, we’ll be able to conclude what is the correlation between the cost and the willingness to purchase.

1. **How many times does a guest attempt to search for a destination that we do not serve for the given airport or date?**

Athena Query Logic:

- Getting all the records for unique searches – VAR 1

Select count(\*) from search\_input

- Getting the records, where seat was unavailable – VAR 2

SELECT count(\*) FROM (SELECT DISTINCT \_id from search\_output WHERE is\_available =False)

- Dividing VAR 2 by VAR 1 will give me the **percentage of bad attempts**