At The Points Analyst, we use data to help our customers travel more, but spend less. Utilizing state-of-the-art search techniques, our AI algorithm searches for First and Business Class award flight redemptions to popular tourist destinations. The results of these flight searches are curated and distributed to our valuable customers.

**Technical Round Introduction**

In the technical round, you will build an airline scraper, which pulls key data (see XL file) from the website of a given airline. While the requirement is not the completion of a “production ready” asset, the asset must be functional.

**General Details**

Airline United Airlines

Data Type Miles (i.e. not Cash)

Key Data See XL File

Time Period Today + 90 Days

Time Limit 1 Week

**Specific Requirements**

Data Inputs

The data inputs to the scraper should be as few as possible. To keep things simple, there are a few data parameters you can “hard code:”

Class Business

Time Period Today + 90 Days

# Stops Non-Stop Only (i.e. no 1+ Stop options)

This would leave two variables as “inputs” to the scraper:

Origin 3-Letter Airport Code (i.e. IAD)

Destination 3-Letter Airport Code (i.e. LHR)

The input mechanism should be able to handle multiple Origin / Destination combinations at a single time (i.e. input the data as a “list,” and all combinations get run).

Airline Scraper

You are free to use any technology to build the scraper, but it must be scalable (i.e. be applicable to other airlines), automated (i.e. limited manual intervention), and efficient (i.e. does not take multiple hours or days to run to completion).

With that said, there are a few baseline requirements:

1. The scraper runs as a standalone dockerized application.
2. The scraper has an exposable API web interface to trigger the scraping process.
3. All external dependencies (for example, Redis cache instance, database, scraping monitoring solution, etc.) must be interconnected using connection strings.
   1. This enables the dockerized solution to be deployed as a microservice in Kubernetes or on separate distinct hosts, which are connected through the network.
4. The scraper deploys on Kubernetes / Azure.
5. The scraper stores data in Azure Blob Storage
6. The scraper loads into GitHub (for review purposes) as a standalone repository.

Bonus Requirement: When deployed on Kubernetes ,or another container platform, the application is horizontally scalable, which enables the use of multiple scraping workers in order to scrape data faster.

Outputs

The scraper / application must produce a single .csv file, which contains the key data (see the XL file) for all origin / destination combinations.

Final Test

One the application is built, there will be a final test to assess functionality.

A list of origin / destination airport codes is provided. The application must take the inputs, scrape the data, and produce the desired output. The airport codes will not be given in advance.

If there are any general questions on the requirements, please feel free to reach out for clarifications. With that said, there will be no guidance on technical build specifics.