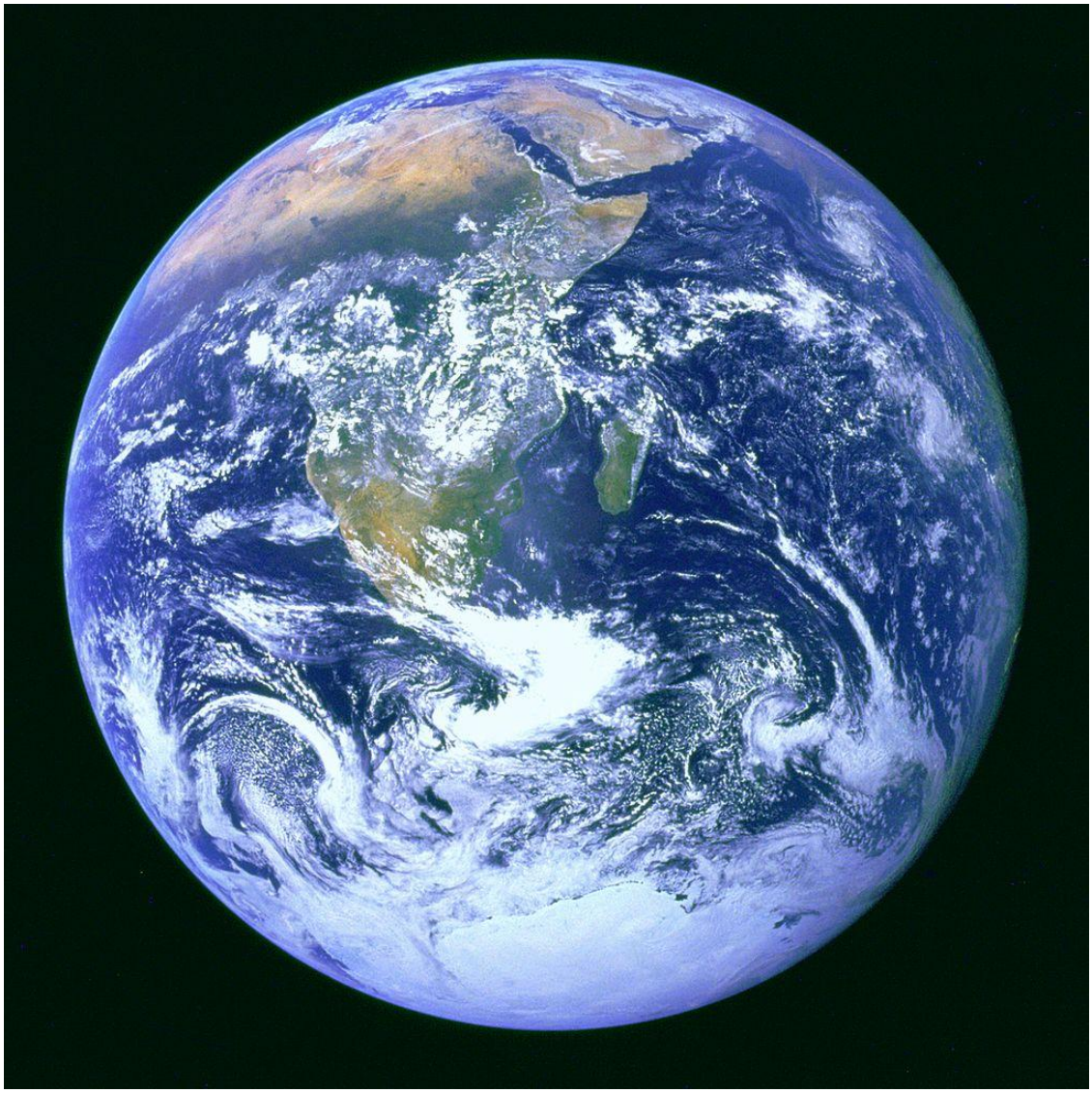

Space Missions Launch Analysis API (SMLA)

Final Project for COE 332

Asher Cura-Portillo - acc4476
Izaac Facundo - imf339
Steven Oh - jo25672

Image Credits: NASA, "Blue Marble."

Motivation

Within the past two decades the growth of the private space industry in the United States has exceeded expectations in its growth. From its role in communications, GPS, and satellite TV to its importance to defense in our country, it's clear that such an important pillar in the United States economy should be better understood by engineering students alike. Compounding this, there are a number of space agencies, apart from NASA, that students are not familiar with. Thus, there is a need for tools to better understand and communicate the activity and economy of entities involved in the space industry– both publicly and privately.

Our project aims to give students methods of better understanding the space industry. To do so, we have created a software code-base with several functionalities capable of returning key pieces of information about space mission launches from as far back as the 1950s. By exploring using our software, students will be able to get a better grasp at modern space launch activity, historical trends, and the current volume of the space economy.

High Level Description

Our API, named the Space Missions Launch Analysis API (SMLA) is a tool capable of organizing and visualizing information on space launches– dating as far back as the space race between the US and the Soviet Union. The domain of the dataset includes the entire globe and is updated with modern space launches as recent as August 2020. Our software makes use of three key technologies, of which we will describe in more detail: Flask, Docker, and Kubernetes. Using these softwares, our API is able to be deployed as a kubernetes cluster, from a docker-compose file, or locally on a user's computer.

By doing so, the user is able to interact with our software to explore more specific pieces of information on space mission launches as well as accessing two methods of visualizing the dataset we have chosen.

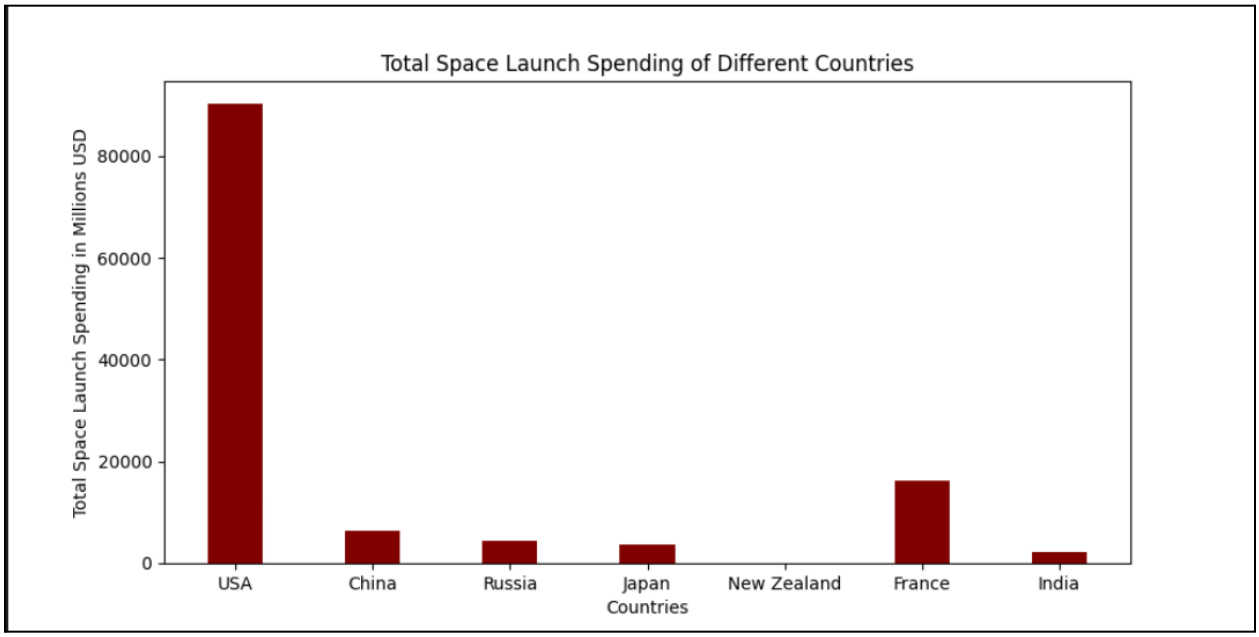

Figure 1: Bar graph depicting total spending of space launches per country.

Above in Figure 1, one such method of visualization is presented. Our API can sum the total cost of spending per country, assuming the spending has been provided, and graphs the cost in the form of a bar graph. Notably, the USA spends a lot more than every other country but this may be due to the dataset lacking information on these other countries or these other countries not making the information publicly available in the first place.

Data Description

The data set contains information of space mission rocket launches from around the world. It is sourced from Kaggle.com [1]. The range of the information goes as far back as the 1950's and still continues to this day. Using this dataset we can create and drop 'mission_launches.csv' into our repositories directory. This is the file that contains all the information that is accessed via the website Kaggle.com. Due to some issues with potentially hosting this data, and the functionality of the Kaggle.com website, the user must download this file themselves in order to operate the repository.

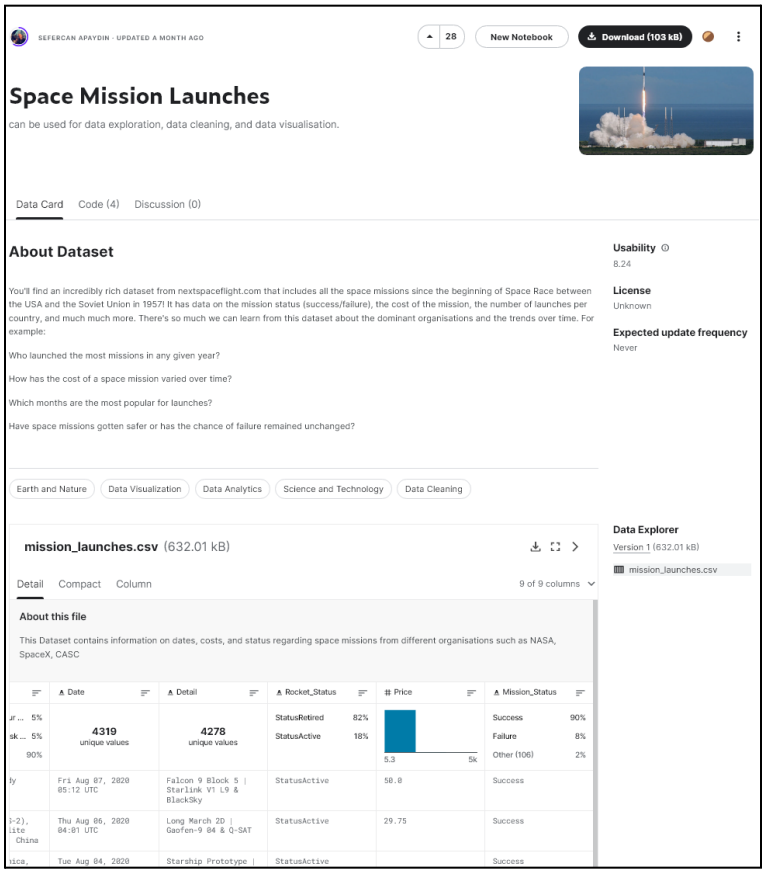

Figure 2: The page of Kaggle.com, with the dataset we use pictured at the bottom.

The information contained includes organizations, locations of launches, date launched, etc. Allowing us to create a map of rocket launches superimposed on a map of the world. As well as identifying who is responsible for which launches. Additionally the approximate price of each launch is sometimes included in each entry of the dataset. Our API uses this information to return the total cost used by each organization. But note, plenty of entries are missing this information as it is often deemed confidential.

Key Technologies

1. **Flask:** The flask library is imported and primarily utilized by 'flask_api.py' within the repository. This python file is responsible for hosting 'front-end' functionalities. In other words, end-users will interface with the functions in this file directly. Through use of the Flask library, we can host our code on a Flask web app server, where the user is then able to access 'endpoints' via the 'curl' command (e.g., 'curl

localhost:5000/jobs). Each endpoint represents a different function, in the previous example the user accessed the 'jobs' endpoint. As a result, the list of all previous jobs is returned to the user's terminal.

2. **Docker:** Docker allows our source code (i.e., 'flask_api.py', 'worker.py', and 'jobs.py') to be containerized. Essentially, Docker will package these files, and their dependencies, into an object called an 'image': a blueprint of everything necessary to run each python file. These files are then uploaded to DockerHub, a website much like GitHub that hosts these images. By doing this, you can allow use of other methods of software installation to greatly speed up development and setup of the software.

3. **Kubernetes:** Kubernetes takes advantage of the aforementioned technologies to quickly create instances of our docker containers. These instances are called pods, or virtual machines that use the 'blueprint' or docker images to act as virtual machines capable of running our code in individual systems (pods). By using several kubernetes files, we can utilize several abilities including: scaling our software deployment up or down, restarting services automatically, abstracting parts of our software, and maintaining modularity.

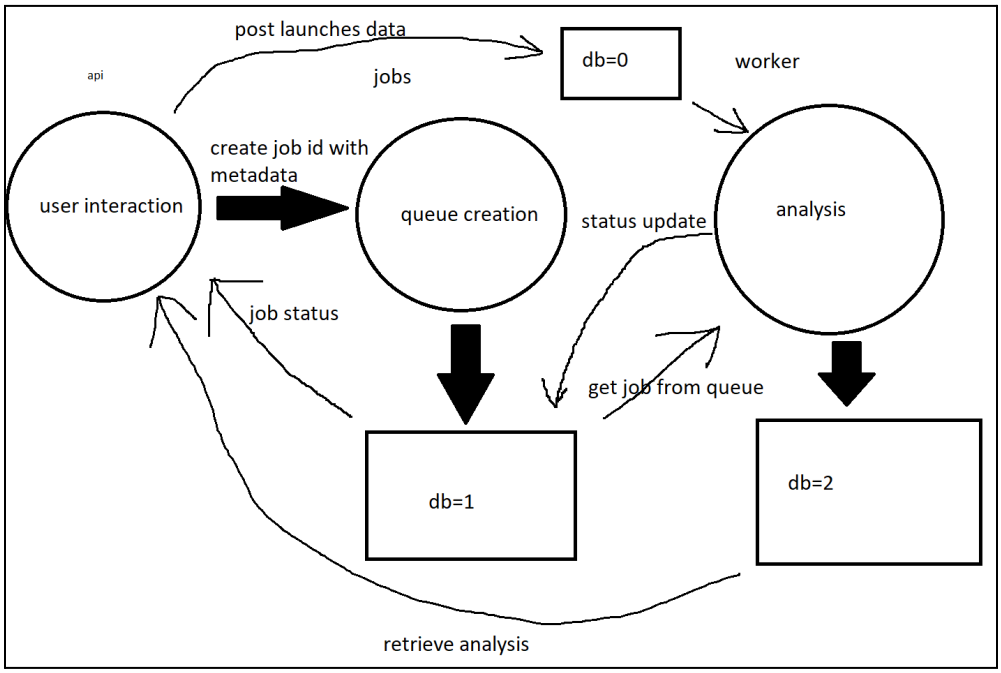

Figure 3: An illustration depicting the full network of kubernetes clusters that make up our software deployment.

By making use of these key technologies our software deployment becomes robust, modular, and scalable. Pictured in Figure 3 is a drawn representation of the network of kubernetes clusters that host several important pieces of our code base, such as: the front-end Flask API, the redis databases, the analysis/worker cluster, and the creation of queues. Aside from the object titled 'user interaction', which represents the cluster which hosts the 'flask_api.py' code, the majority of the systems are abstracted from the user.

List of Endpoints

Here is a list of endpoint routes contained in the 'flask_api.py' file. These are the commands that are to be accessed by the user on the front-end of our code base. Note, that although there are only 3 endpoints listed, there are multiple ways to use each endpoint.

1. **'/data':** This endpoint is related to the data of the mission launches stored in-memory. It directly accesses the information on the space launches contained within the Redis databases. Thus, it affects the data that is accessed by every file in the codebase. This endpoint can be used in conjunction with the 'GET' command to return the information currently stored in the database to the user. This is useful if the user needs to view what is currently being stored in-memory, or to check if there is any information being stored at all. The user can also use the 'DELETE' command on this endpoint to clear the stored memory. In the event that the user does not find any data in the database, they can use the 'POST' command to update the database with the .csv file contained in the repository.
2. **'/jobs':** This endpoint is related to the queuing of 'jobs' or tasks related to analyzing or visualizing the data (e.g., creating the map, returning a list of all organizations). Using the 'GET' command on this endpoint will return a list of python dictionary objects containing information on all previous jobs the user has created.
    a. **/jobs/<string:JOB_ID>:** If the user adds another '/' and types out the ID of a previous job after the slash, then the dictionary object of a specific job will be returned. Uses the 'GET' command.
    b. **/jobs/<string:ROUTE>:** If the user adds a string containing the name of a function, then a job will be created and the result will be stored in Redis.
    c. **/jobs/clear:** If the user uses the 'DELETE' command on this endpoint, the current list of jobs will be cleared.

3. **'/help':** This endpoint is accessed via the 'GET' command. Users can access it by curling the IP address of the Flask server. A text message containing information on usage of all the endpoints will then be returned to the user.

Usage

Here a few example queries:

Post a job to the database:

1. First make sure the Kubernetes cluster is set up and Docker is running:

For kubernetes:
Make sure to use this program to start all of the kubernetes elements:

```
>>> kubectl apply -f  <name of the deployment, service, ingress, or
pvc>
```

For Docker:
To get the docker-compose to run in the background:

```
>>> docker-compose up -d
```

2. Make sure to post the data before querying to get actual results. Else there will be an error message waiting.

```
>>> curl -X POST localhost:5000/data
```

3. Next, add a job to the queue using this query:

```
>>> curl -X POST localhost:5000/jobs/'<specific job>'
```

4. Finally, find and retrieve the jobs list (which can be done through a browser):

   http://spacemissionanalysis.coe332.tacc.cloud/jobs

Example:


```
[
  {
    "id": "e340f255-853e-4102-970a-e994c73ef3bc",
    "route": "get_orgs",
    "status": "incompleted"
  },
  {
    "id": "2a150042-0e8c-401f-8747-e7c811fbed42",
    "route": "map_of_launches",
    "status": "in progress"
  },
  {
    "id": "7c0ff2af-70f3-4fc5-b8bf-f318c50c2148",
    "route": "test",
    "status": "incompleted"
  },
  {
    "id": "03991f72-48d7-482e-b125-5d4ac81ba948",
    "route": "test",
    "status": "incompleted"
  },
  {
    "id": "f06119df-9f00-45f6-9a20-2f1d8bbe2169",
    "route": "get_rockets_by_org-ULA",
    "status": "completed"
  },
  {
    "id": "3b3d74a2-97cc-4ff5-a7cc-e7c041188871",
    "route": "get_orgs",
    "status": "completed"
  }
]
```


Other information on how to use this application and its setup can be found in the github repository created:

https://github.com/jaeestee/Space-Mission-Analysis-API

Ethical and Professional Responsibilities

In software development and engineering, ethical and professional responsibilities are essential aspects that should be considered throughout the development process. In the case of our project, the Space Missions Launch Analysis API, we recognize the importance of upholding ethical and professional standards to ensure that our codebase is not only useful but also reliable and trustworthy.

One ethical responsibility that we must adhere to is the importance of citing data sources. Our codebase relies on the use of a dataset sourced from Kaggle.com, which contains information on space mission rocket launches from around the world. It is crucial that we give credit to the original data source to acknowledge their contribution to our project. Moreover, citing data sources allows users to verify the accuracy and quality of the data used in our application.

Another ethical responsibility that we must consider is the privacy and confidentiality of the data used in our project. As mentioned, the approximate price of each launch is sometimes included in the dataset, but many entries are missing this information as it is often considered confidential.

Similarly, the licensing of this dataset is ambiguous, thus we have chosen to assume the original creators would not allow us to host the dataset ourselves. As a result the user must download the dataset from Kaggle.com and insert into the repository themselves.

Citations

[1] S. Apaydin, "Space Mission Launches". *Kaggle.com*. Accessed April 25th, 2023 from https://www.kaggle.com/datasets/sefercanapaydn/mission-launches?resource=download