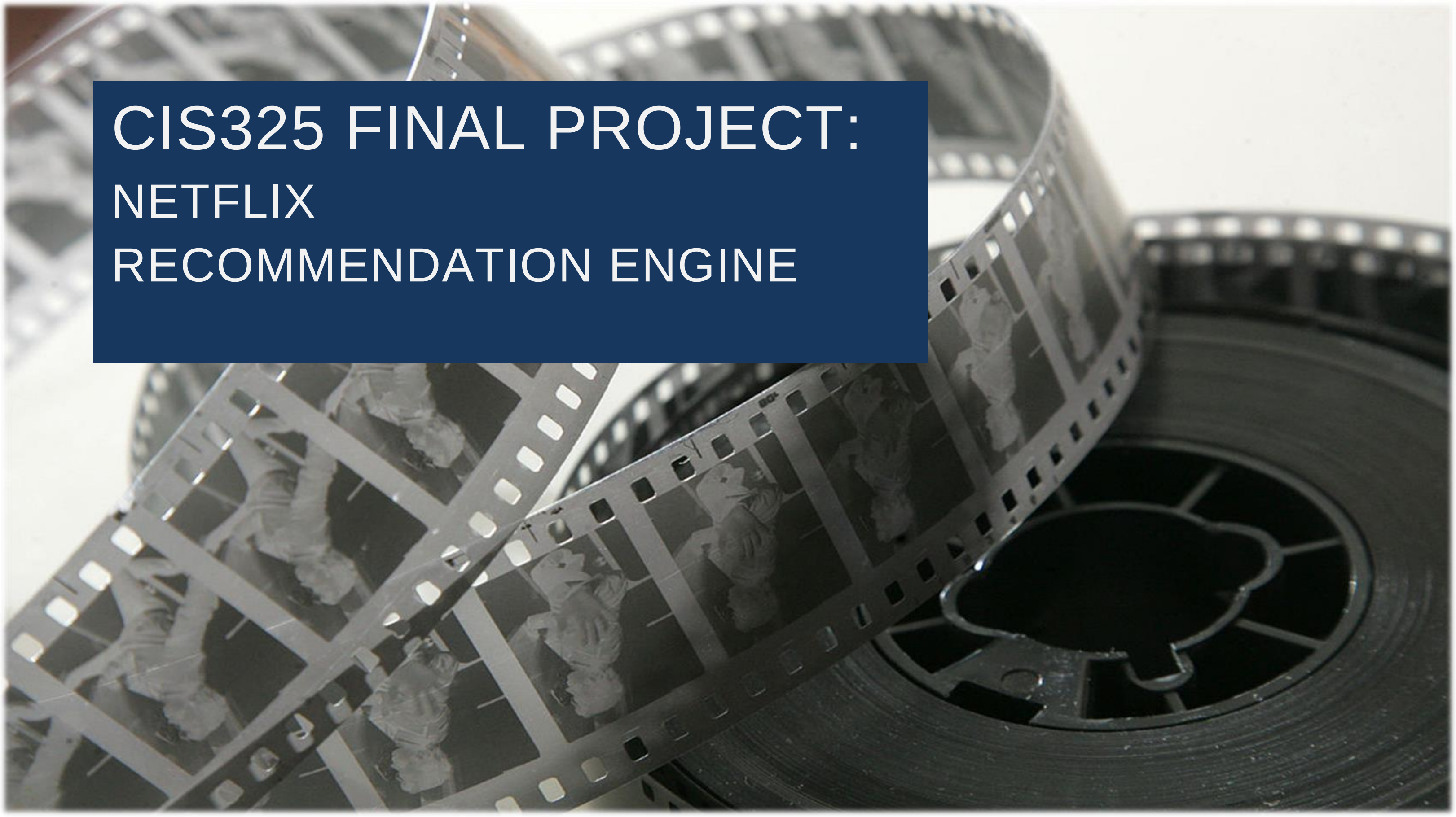
CIS325 FINAL PROJECT:

NETFLIX RECOMMENDATION ENGINE



The goal of this project is to simulate an enterprise level data science challenge by engaging a small team of students to conduct an end-to-end process on a chosen dataset. This includes building and deploying a machine learning model as a web service which can be demonstrated as a client-side application.

The selected dataset was sourced from Kaggle[1] and based on publicly available Netflix data from 2021, containing information on 8800+ steaming movies and TV shows at the time. This robust data features details such as movie title, genre, director, cast, release year, country of origin and description of content.

Our model will allow a user to input a movie description of what might be available on Netflix and receive a recommendation for interesting titles, much like what a consumer would experience when navigating the platform on their device.

[1] Online Kaggle source: https://www.kaggle.com/datasets/shivamb/netflix-shows

GETTING STARTED

DEPENDENCIES:

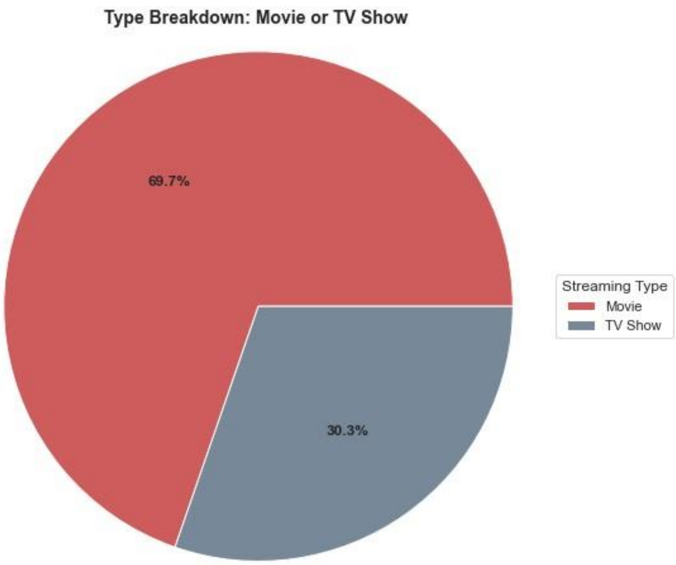
- Python 3
- Scikit-Learn
- Pandas/Numpy
- Seaborn/Matplotlib
- Current Web Browser
- Jupyter Notebook
- MLflow
- Tensorflow
- Pickle
- Microsoft Azure Portal
- Flask, Flasgger APIs
- Docker/CLI

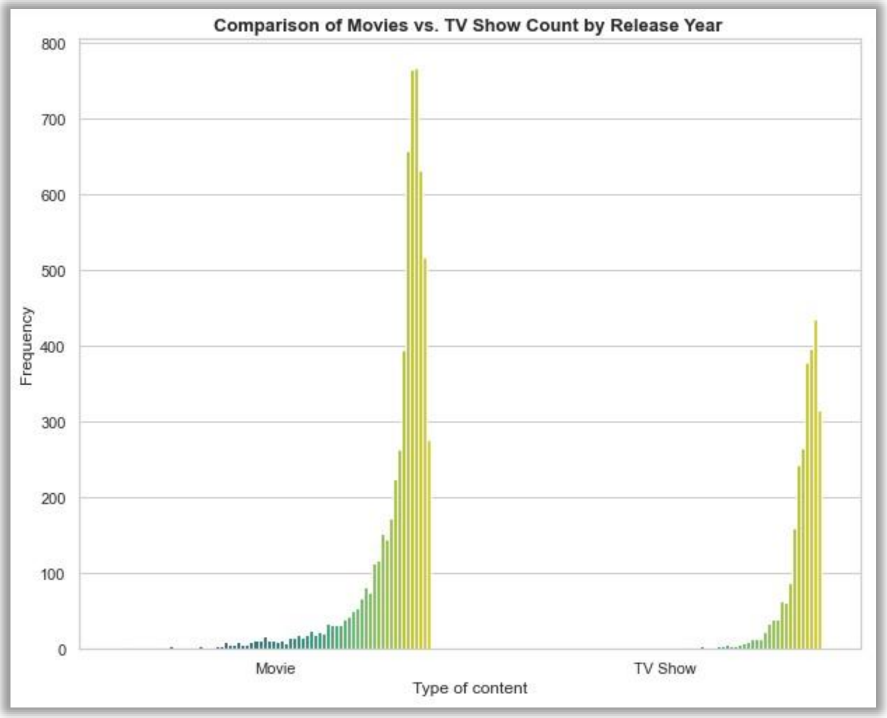
INSTALLATION RESOURCES:

- Install all relevant Python packages: www.anaconda.com/products/individual (such as Scikit-Learn)
- Pandas: https://pandas.pydata.org/pandas-docs/stable/getting_started/install.html
- Install MLFlow package: www.mlflow.org/docs/latest/quickstart.html#installing-mlflow
- Docker overview: https://docs.docker.com/desktop/
- Container registry setup: https://docs.microsoft.com/en-us/azure/container-registry/container-registry-get-started-portal#create-a-container-registry
- Azure Kubernetes Service: https://azure.microsoft.com/en-us/overview/kubernetes-getting-started/

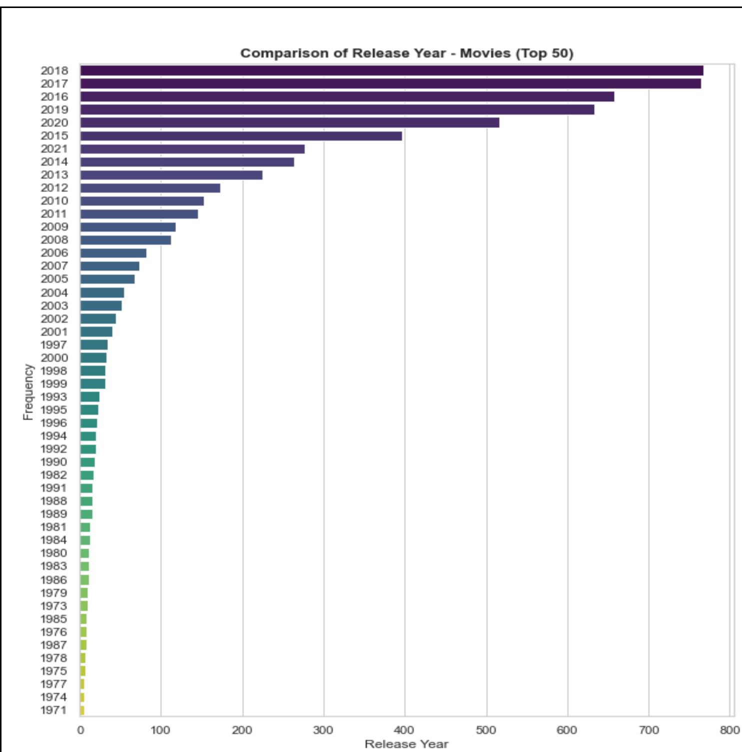
EXPLORATORY DATA ANALYSIS

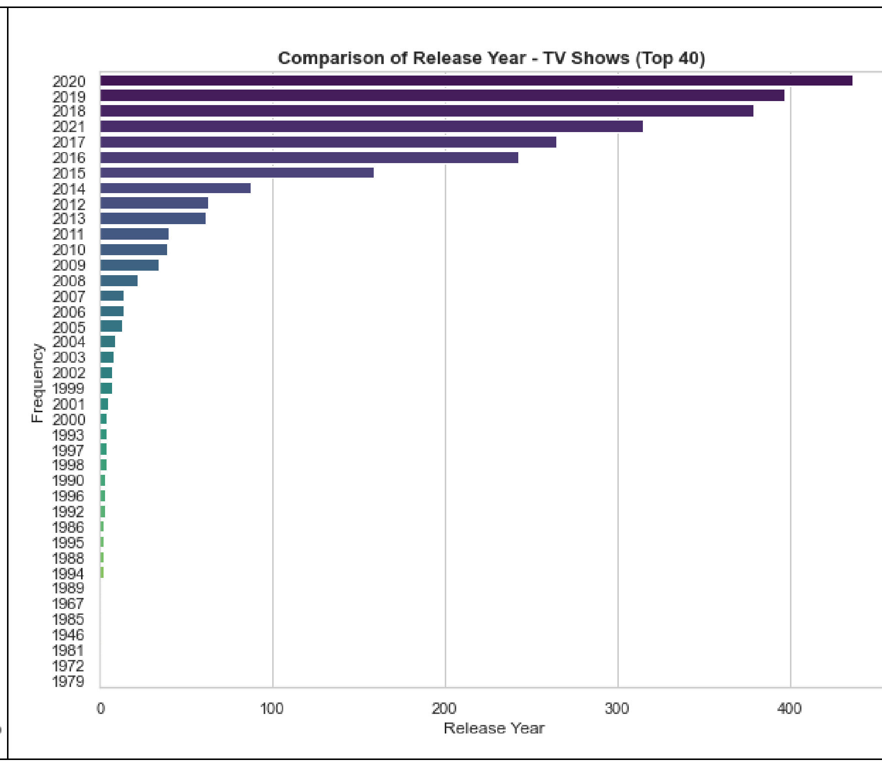
It is important to note that the bulk of the data pertains to movie content. This could mean that any recommendation engine could offer better accuracy when searching for similar movies since there is a greater pool of data for that type.



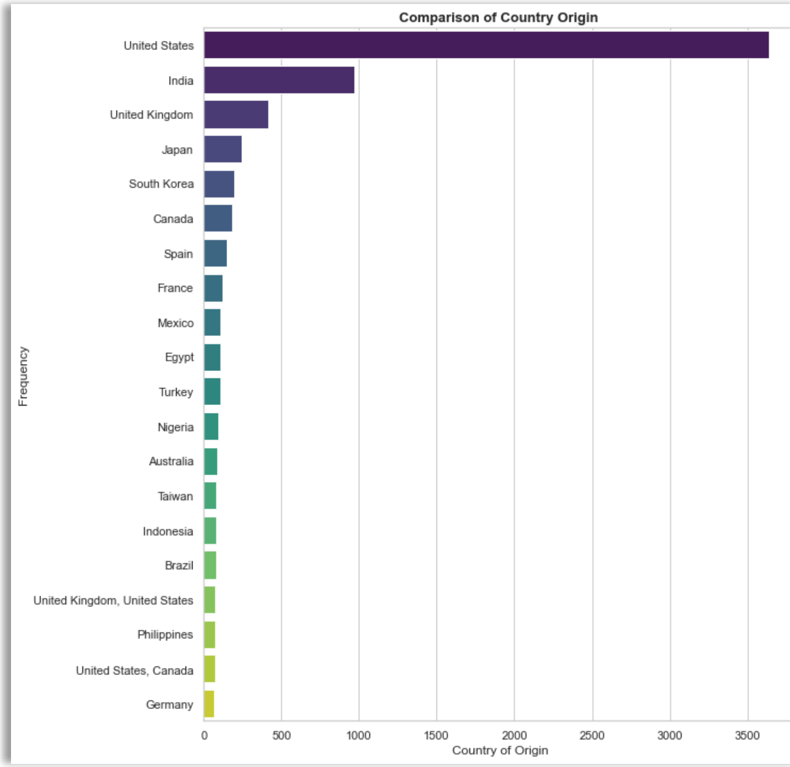


There is definitely an increase in streaming content as time passes except for a sharp decline in the past two years due to the pandemic, however the general curve in the frequency count increase seems very similar between movies and TV shows, even though movies account for far more content overall. There was slightly less of a drop off in TV shows however, as they are easier to produce with smaller production needs.

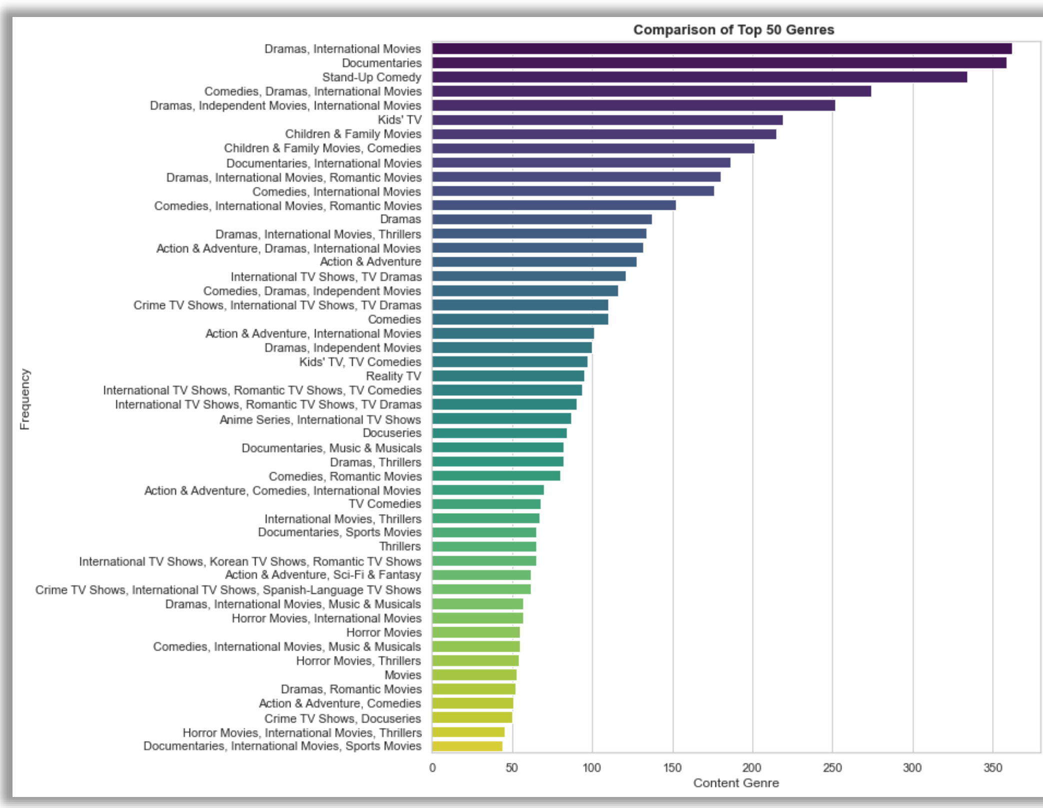




There is a bit of a discrepancy as more TV shows were released from 2018-2020 than 2021, likely because the data for 2021 is incomplete. More movies were released from 2016-2020 than 2021, also likely because the data for 2021 is not complete. However clearly less movies were produced in 2020 than prior years-- no doubt due to the global pandemic and filming shut downs.

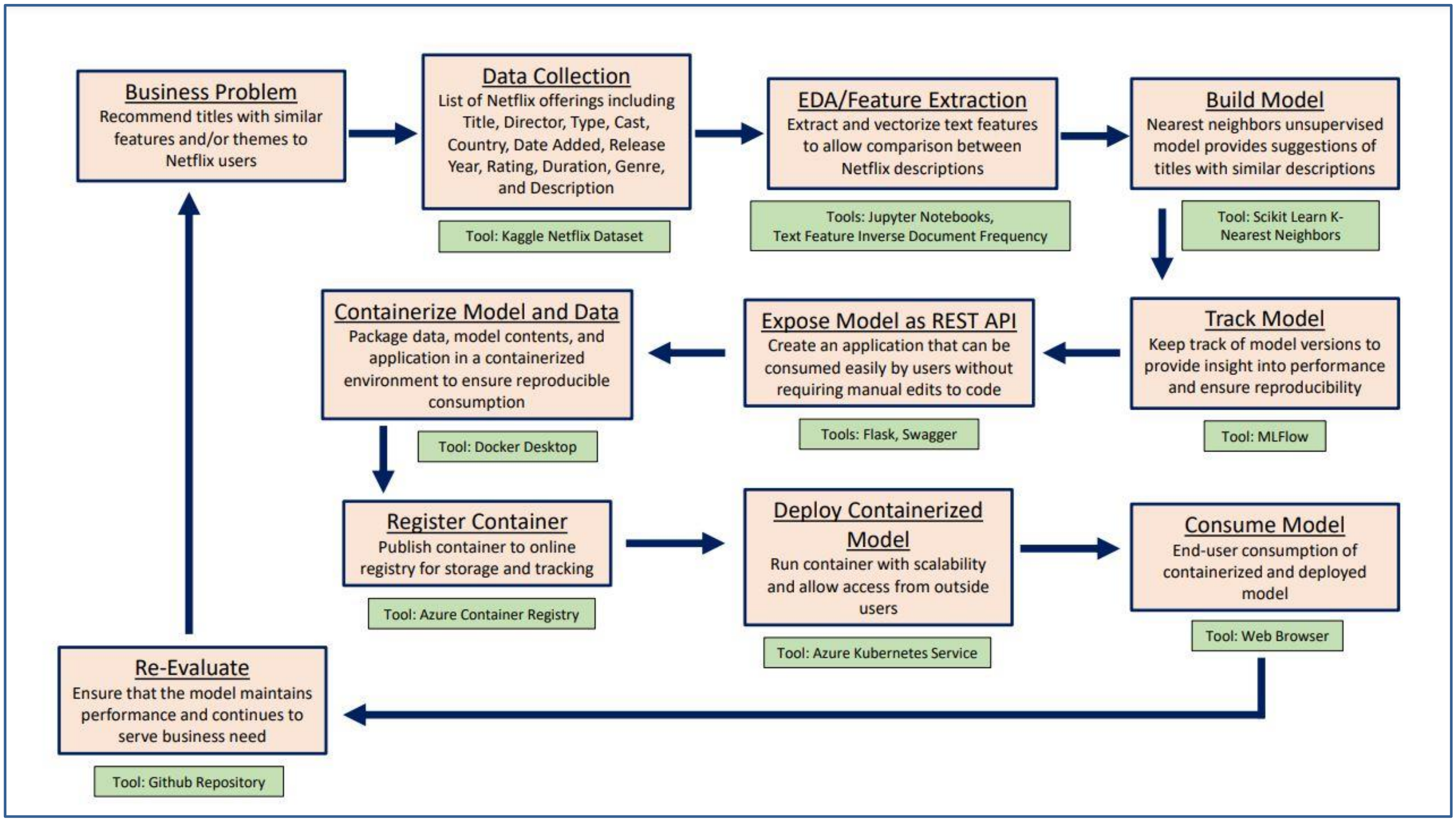


As expected, the US, UK and India (Bollywood) are the top 3 content producers. Countries with well-funded arts programs are also in the top 10 (Canada, France, Japan, South Korea) however I am surprised that Nigeria is not higher up in the list since they have Nollywood- but that just means Netflix is not releasing much of their content. It's also surprising that Spanish-speaking countries are not better represented since there are a lot of Spanish-speaking consumers for streaming content.



Clearly dramas of all kinds, comedies, romance, documentaries and international films are highly represented. These categories may perform best in the recommendation engine since the data pool for them will be large.

PROJECT DESIGN FLOWCHART

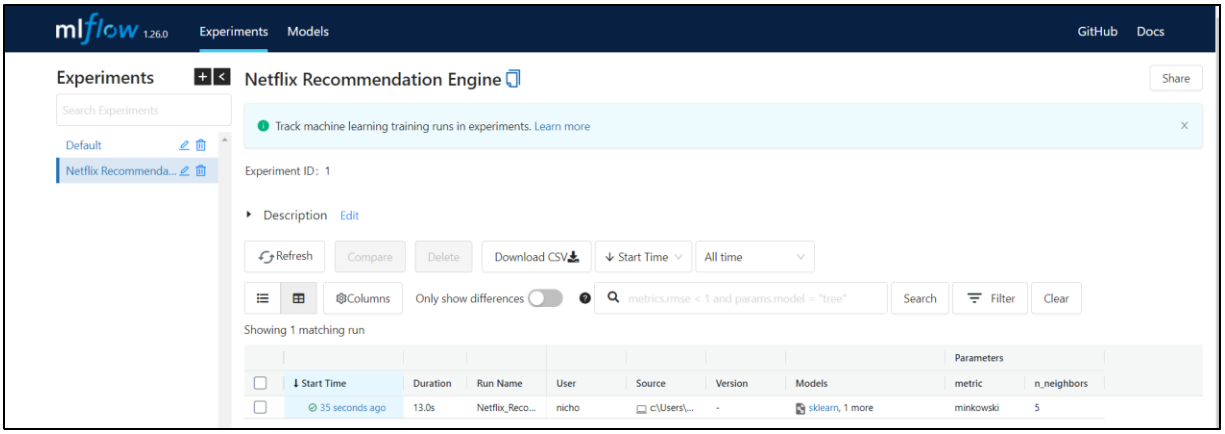


MODEL CREATION

As a first step, the Description column containing the text of the movie synopsis was vectorized and converted into a sparse matrix which contains the movie titles on the index and word or phrase vectors as column features. While cosine similarity and K-means clustering were explored as options to create the recommendation engine, ultimately a K-Nearest Neighbors model was used to expose a model via REST API. A model creation function was employed to search the matrix for the 5 ‘nearest neighbors’ and provide similar movie or show titles to the end user.

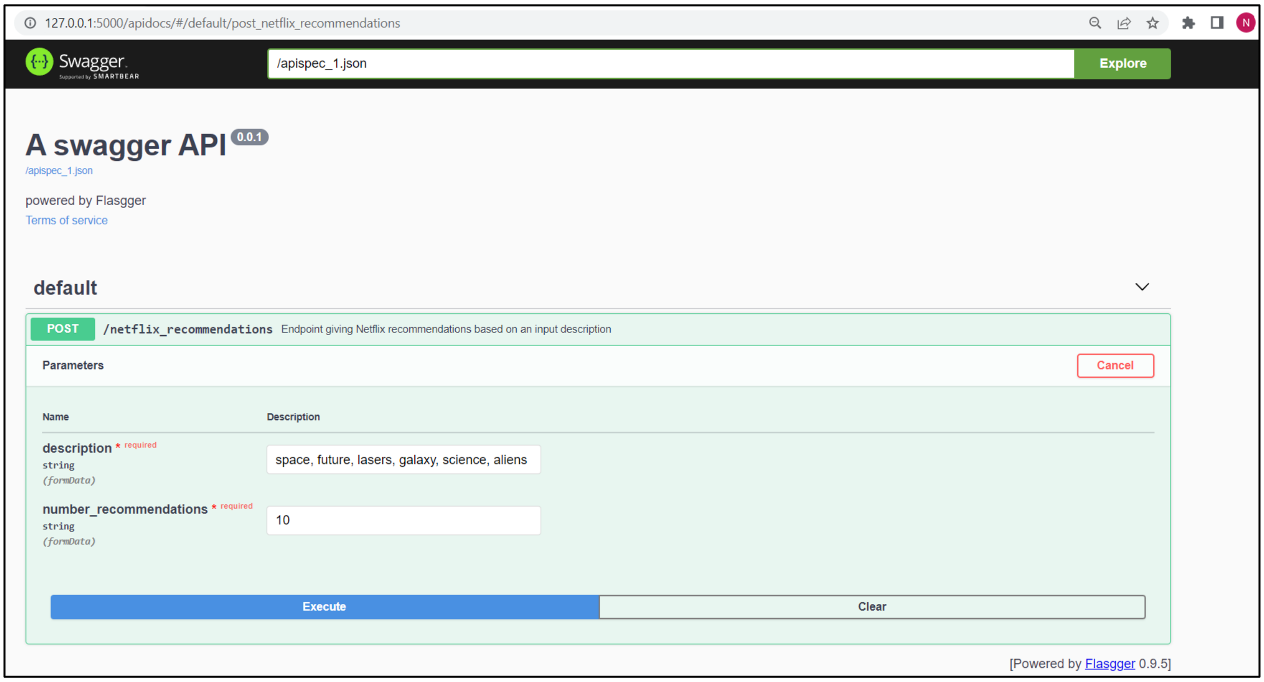
MLFLOW DASHBOARD

Once a new experiment has been created and MLFlow tracking has been established, the MLFlow portal will list any model runs (successful or not) as seen below in the experiment tracking interface.

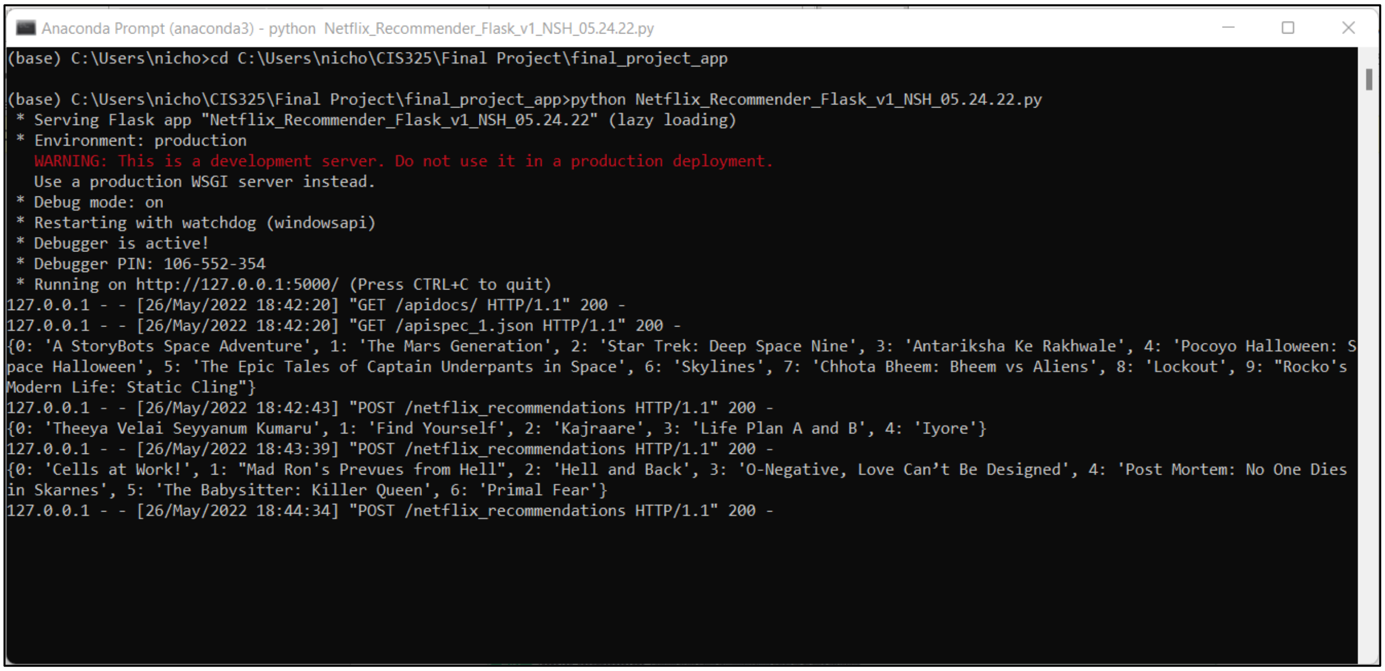


SWAGGER/REST API TESTING

Once a model is successfully tracked and saved via MLFlow, it should be tested locally by creating a Flask python file (.py) and exposed on a local port (such as 5000) using the Swagger API, entering the proper input to test the expected output.



In this case, the Flask app file was programmed to print output (the requested amount of recommended titles) to the command line interface. Three different sets of inputs were tested and the movie recommendations can be seen printed to the CLI below.

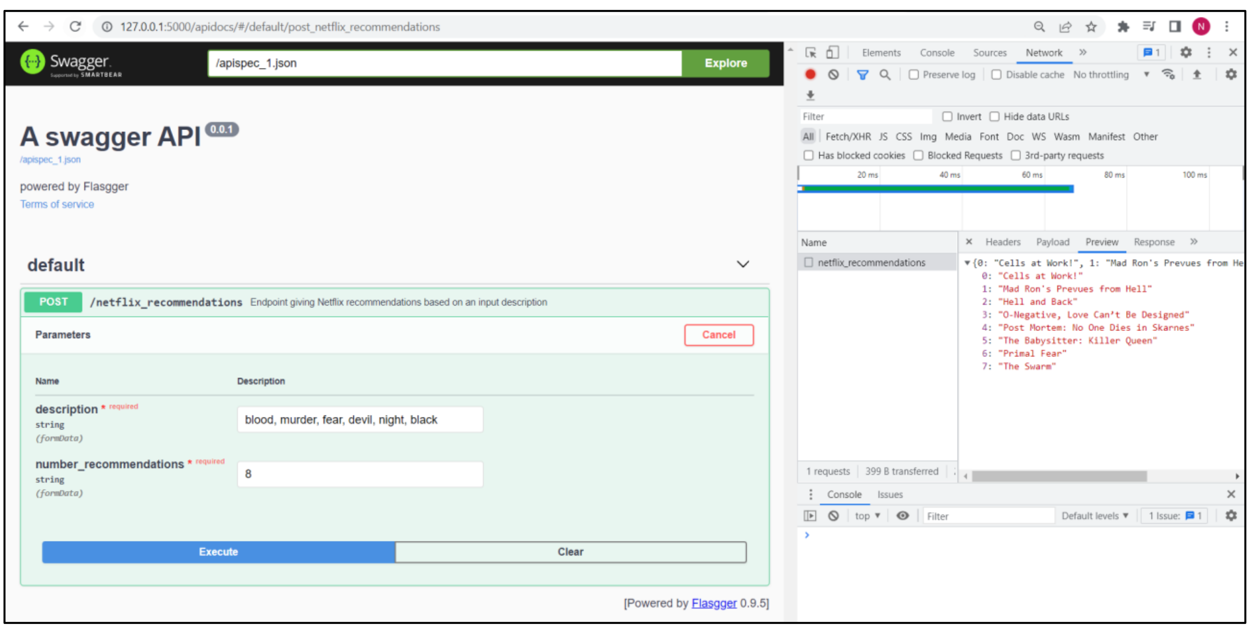


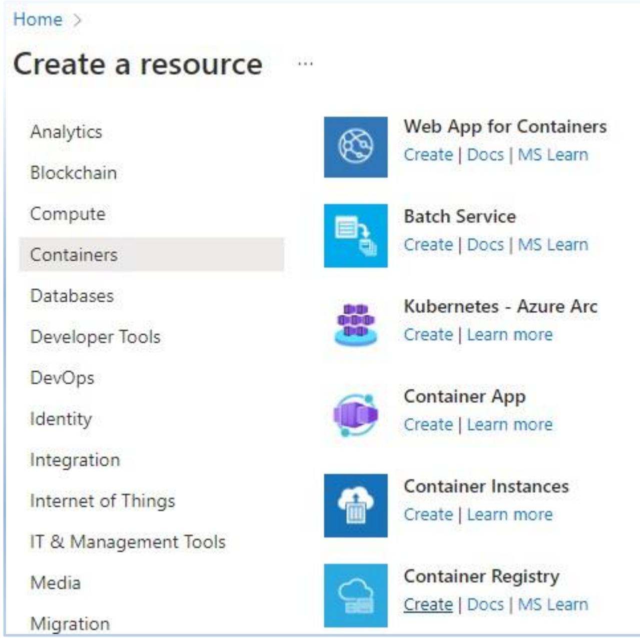
CONTAINERIZATION WITH DOCKER

The model and all of the relevant data and dependencies are then containerized using Docker.

```
Command Prompt - docker  run -p 5000:5000 netflix-recommender-api

C:\Users\nicho\CIS325\Final Project>docker build -t netflix-recommender-api .
[+] Building 54.9s (9/9) FINISHED
 => [internal] load build definition from Dockerfile                                                           0.0s
 => => transferring dockerfile: 32B                                                                            0.0s
 => [internal] load .dockerignore                                                                              0.0s
 => => transferring context: 2B                                                                                0.0s
 => [internal] load metadata for docker.io/library/python:3.8.8                                               2.5s
 => [internal] load build context                                                                              0.0s
 => => transferring context: 3.16kB                                                                            0.0s
 => CACHED [1/4] FROM docker.io/library/python:3.8.8@sha256:e84c219fe873ab169551469f32b57facf7d7baded941ccf0cbcc5  0.0s
 => [2/4] COPY ./final_project_app /usr/local/python/                                                          0.0s
 => [3/4] WORKDIR /usr/local/python                                                                            0.0s
 => [4/4] RUN pip install -r requirements.txt                                                                 49.7s
 => exporting to image                                                                                         2.5s
 => => exporting layers                                                                                        2.5s
 => => writing image sha256:cf888d8578207bd54cf68383a65dee1597fc92ba643ccaaff58e5435b31f7058                   0.0s
 => => naming to docker.io/library/netflix-recommender-api                                                     0.0s

Use 'docker scan' to run Snyk tests against images to find vulnerabilities and learn how to fix them

C:\Users\nicho\CIS325\Final Project>docker run -p 5000:5000 netflix-recommender-api
 * Serving Flask app 'Netflix_Recommender_Flask' (lazy loading)
 * Environment: production
   WARNING: This is a development server. Do not use it in a production deployment.
   Use a production WSGI server instead.
 * Debug mode: off
 * Running on all addresses (0.0.0.0)
   WARNING: This is a development server. Do not use it in a production deployment.
 * Running on http://127.0.0.1:5000
 * Running on http://172.17.0.2:5000 (Press CTRL+C to quit)
172.17.0.1 - - [27/May/2022 01:44:17] "GET /apidocs/ HTTP/1.1" 200 -
172.17.0.1 - - [27/May/2022 01:44:17] "GET /apispec_1.json HTTP/1.1" 200 -
172.17.0.1 - - [27/May/2022 01:44:17] "GET /flasgger_static/favicon-32x32.png HTTP/1.1" 200 -
```

An example of consuming Flask app with a Docker container:

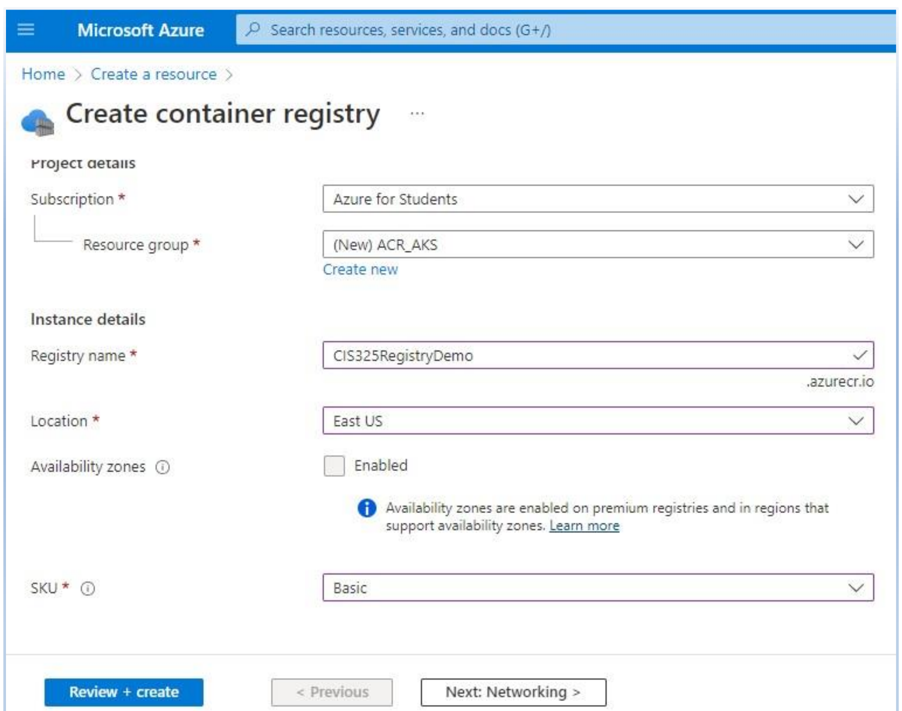




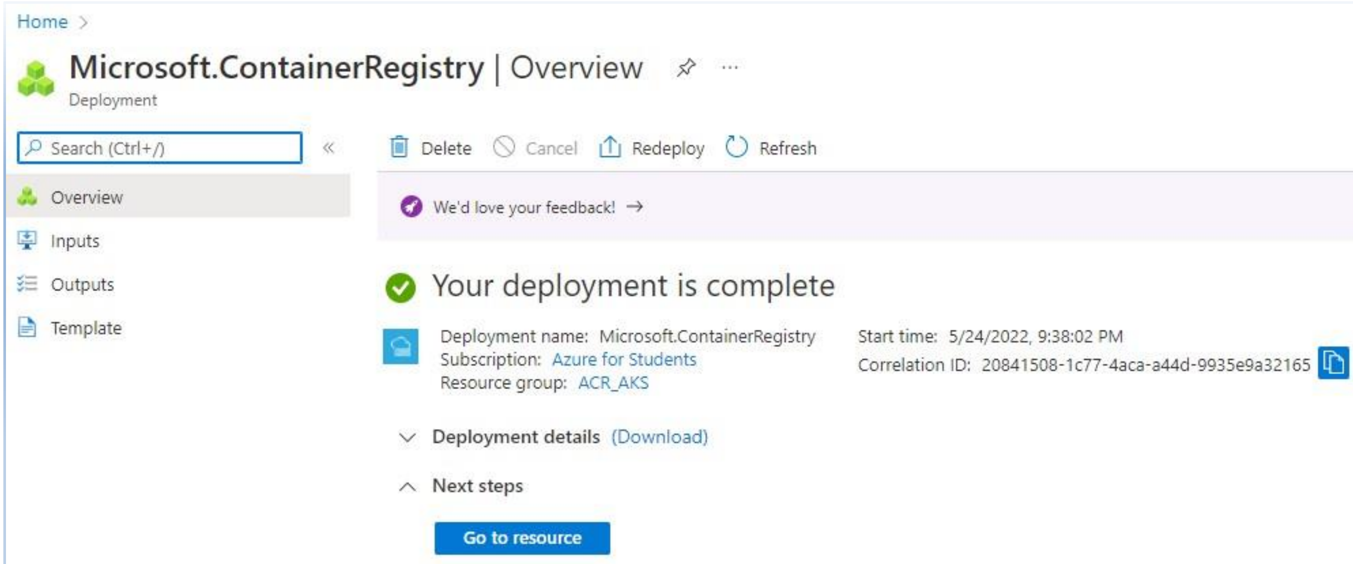
Creating a container registry on ML Azure is as simple as creating a new virtual machine. The first step is to log into the portal and choosing the option to create a new resource, and then searching the options for “Container Registry”.

Select ‘create’ to launch the next step.

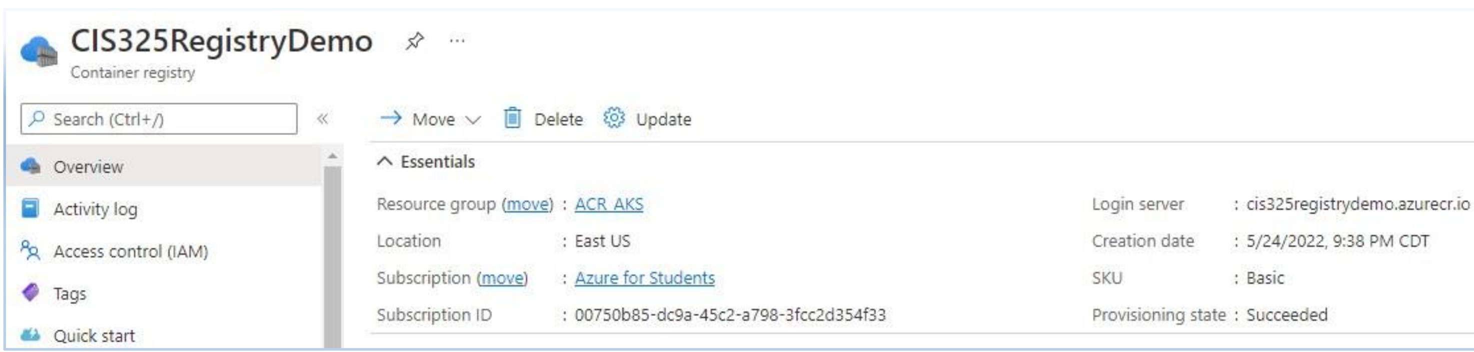
The first and most important step of creating the registry is to choose the subscription option, name the new resource group, name the registry, choose the region location and a Basic SKU for our purposes. For the rest of the screens, the default options will suffice.



Review and create the resource. The final screen will confirm that deployment is complete.



After the Container Registry has been successfully deployed, on the resource home screen you will be able to view valuable information for future use as seen below:



It can also be viewed on the command line interface by invoking the **az acr list** command as shown below.

```
C:\Users\nicho>az aks get-credentials --resource-group CIS325FinalProject --name cis325akc
Merged "cis325akc" as current context in C:\Users\nicho\.kube\config

C:\Users\nicho>kubectl get nodes
NAME                                STATUS   ROLES   AGE   VERSION
aks-agentpool-51973739-vmss000000   Ready    agent   59m   v1.22.6

C:\Users\nicho>az acr list --resource-group CIS325FinalProject --query "[].{acrLoginServer:loginServer}"
[
  {
    "acrLoginServer": "cis325acr.azurecr.io"
  }
]
```

CONTAINER REGISTRATION WITH KUBERNETES

The following steps show how to build and register an image in the previously created container registry.

First, log into Azure via the CLI and run the command **az login**.

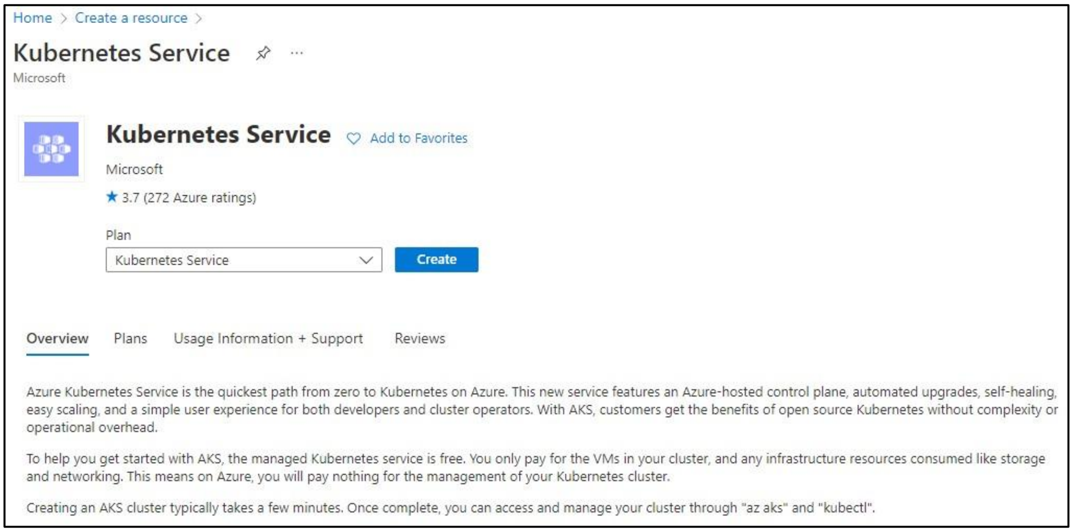
Be sure to have the exact Container Registry name saved, which can be found on the Azure portal, and have the Docker 'daemon' running on your machine.

To authenticate to the Azure ACR, on the command line interface fun the following command:
az acr login <registry_name> --name

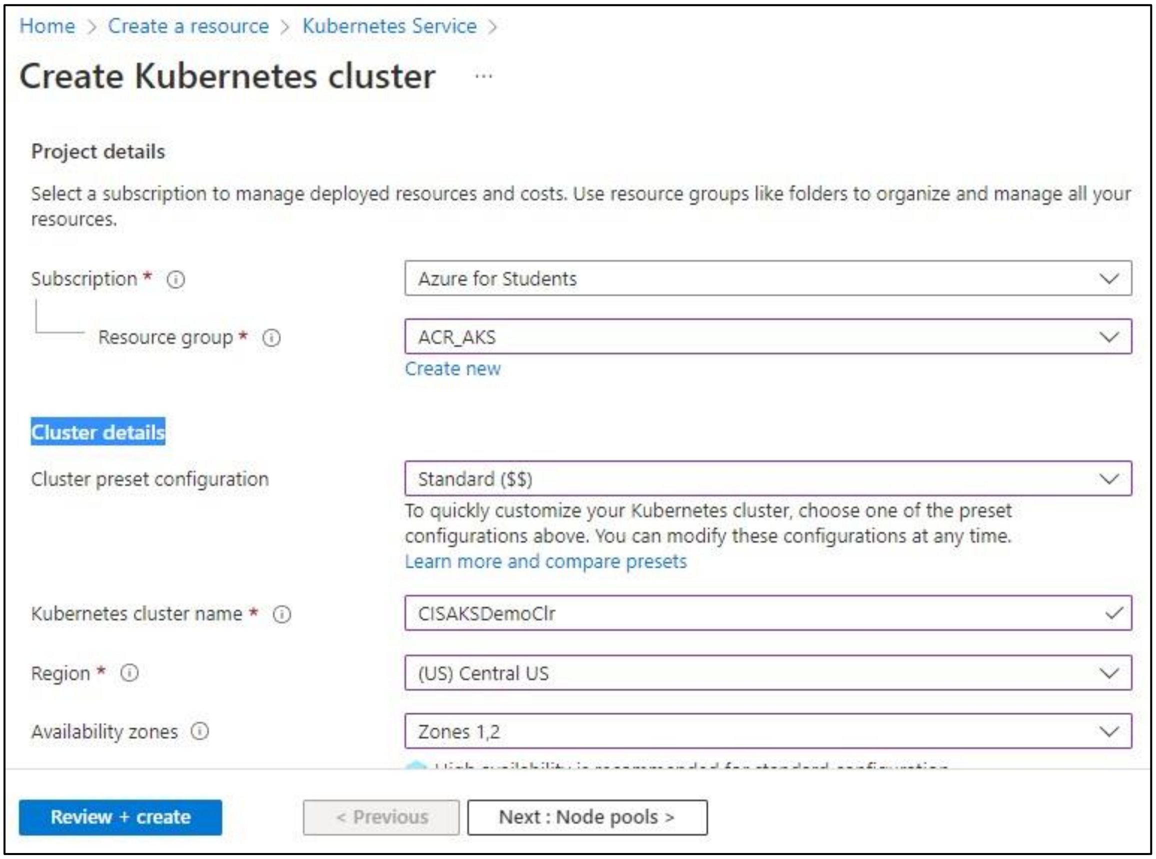
Be sure you are working in the proper directory where files are stored. If necessary change to where your Dockerfire is located.

Now type the following command to build and register your docker image with the ACR:
az acr build --image myimage:v1 --registry <registry_name> --file Dockerfile .

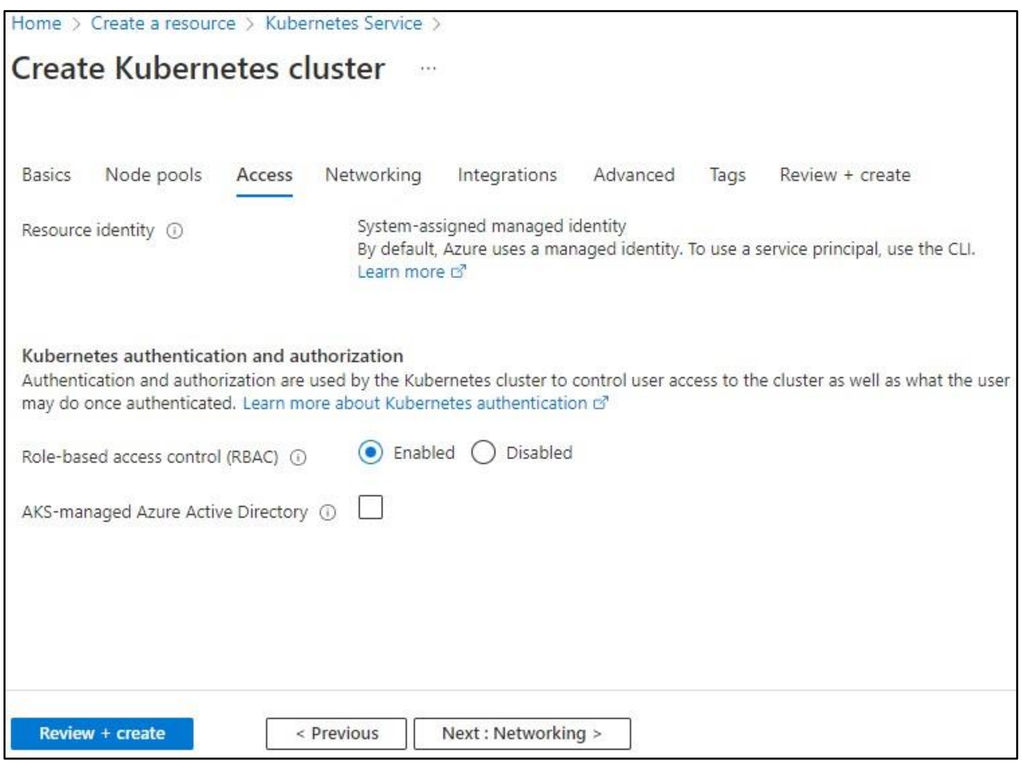
Now, the Kubernetes cluster must be created. This is similar to creating a VM or the Container registry. Search for a Microsoft Kubernetes Service under available resources and select 'Create'.



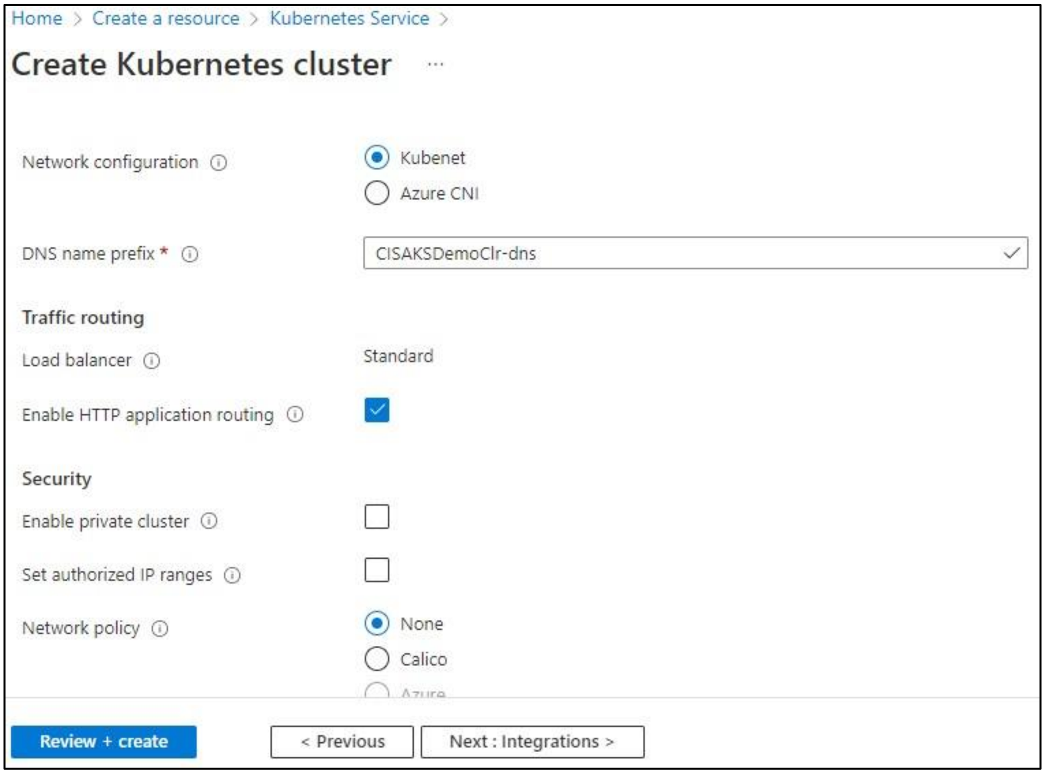
It's important to note the proper criteria to enter or select on each screen. On the basics tab, choose the proper resource group with the container registry, use a Standard configuration, name the Cluster, choose the region, available zones and node sizes.



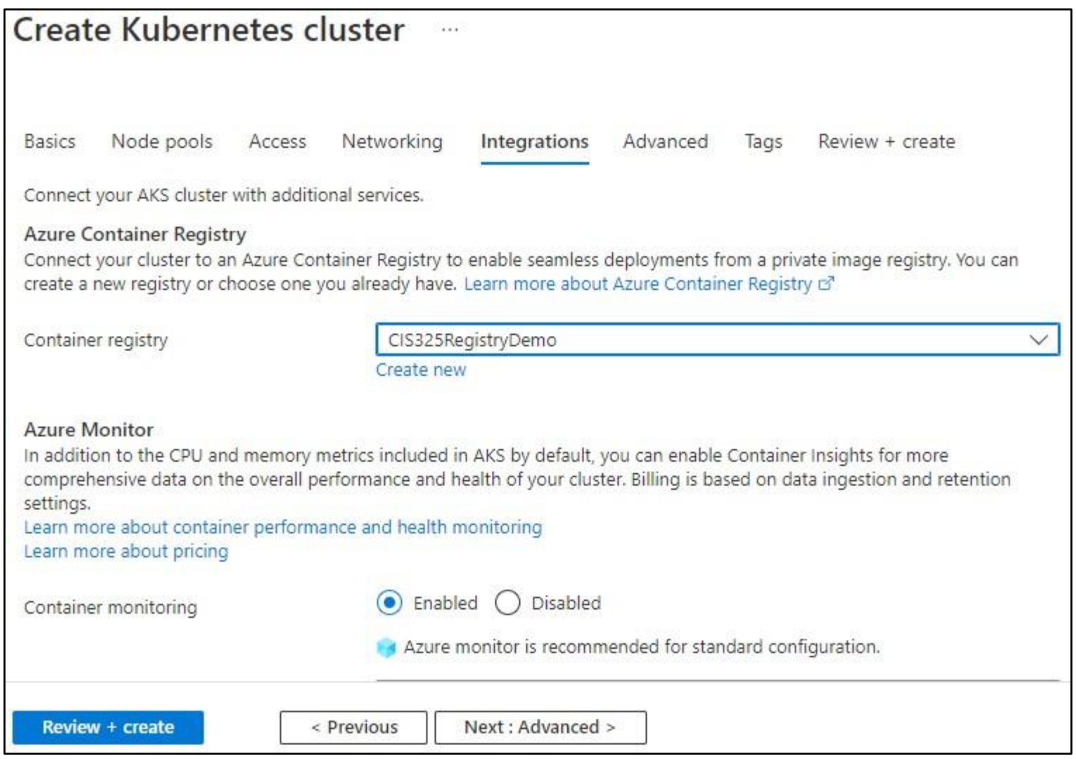
The Resource identity should be System-assigned managed identity.



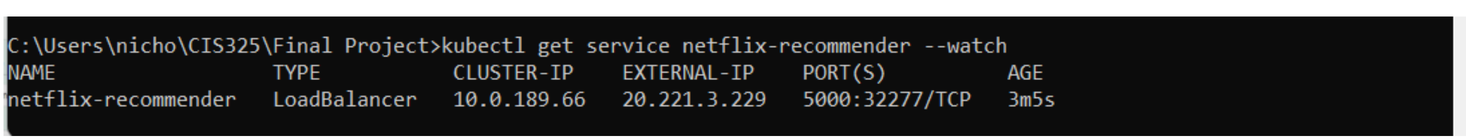
Make sure that you check yes to Enable HTTP application routing.



Select the previously created Container Registry and leave Container monitoring enabled.

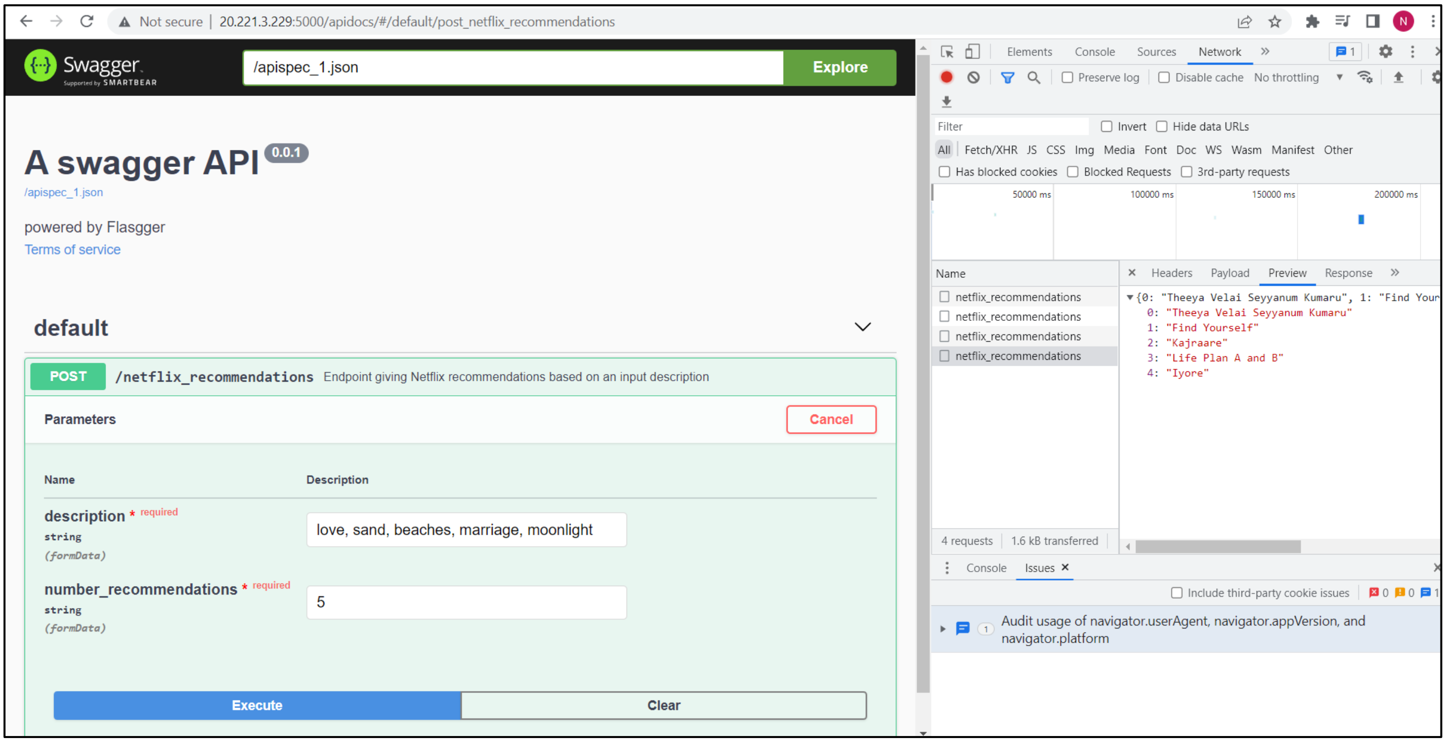


Now you can review and create the cluster. Here is what the deployment confirmation will look like on command line interface:



END USER DEPLOYMENT & TESTING

Now the model can be publicly deployed as a web service. Using the “External IP”, you can log on to the user interface on any web browser. In this example, our public model testing IP is **20.221.3.220**.



As you can see above, a description was provided into the API with phrases and 5 titles were ‘recommended’ by the engine.

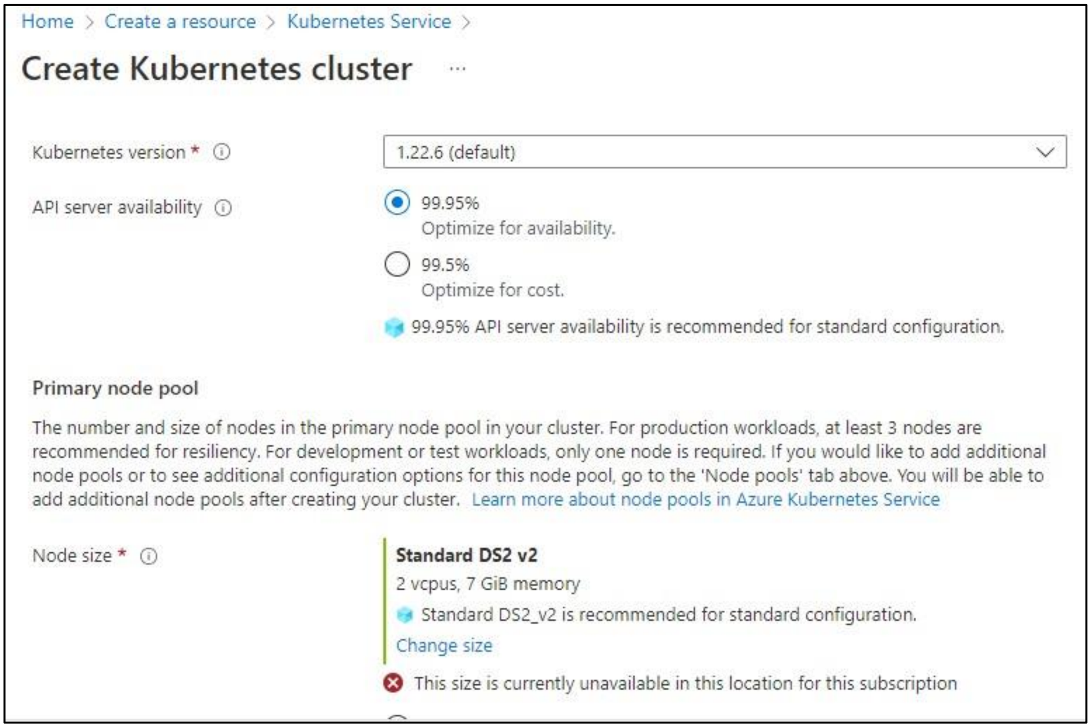
ALTERNATE RECOMMENDATION TECHNIQUES

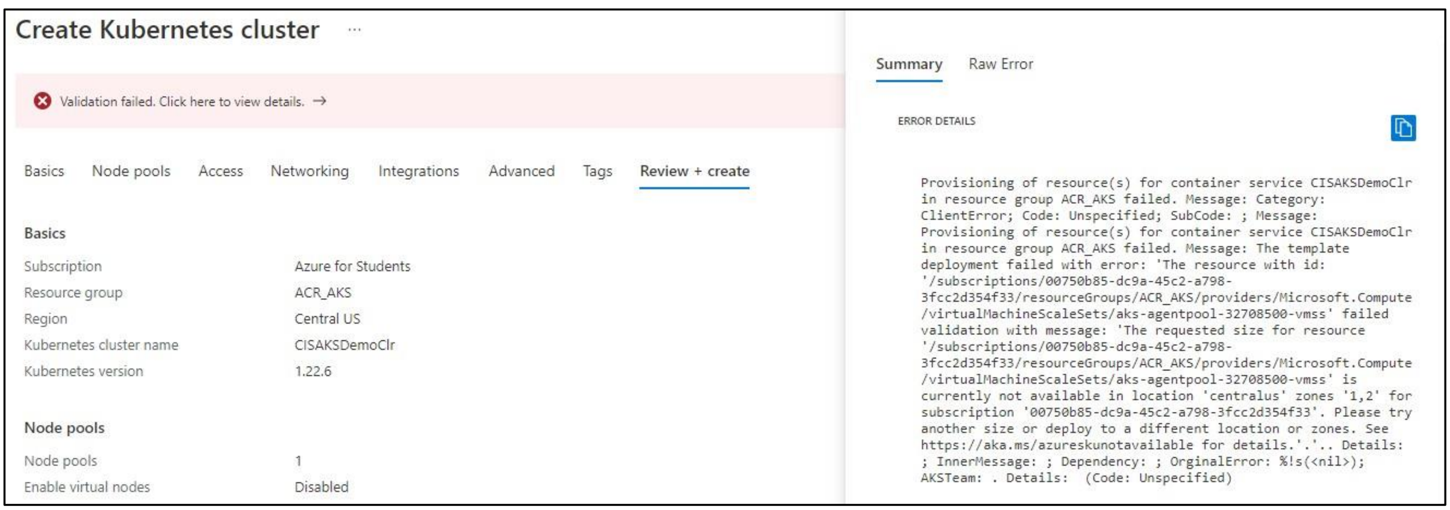
An alternate method of creating a web-based recommendation engine was explored using cosine similarity to score and sort the spare matrix vectors for each title and then a recommendation function used to retrieve the 10 most similar titles. In this case the user can input just the title of the movie or show they liked and a list of similar titles would be returned. This proved to be a viable alternative method, with the majority of provided titles having similar genres, character and plot points. However, as this was not a true machine learning model with unique parameters that can be saved such as KNN or KMeans, it was not used for the API.

```
get_similar_titles('Midnight Mass')

7257    La Rosa de Guadalupe
1906                    Evil
3074                 Save Me
2971                Diablero
4268                Bird Box
7025              Hungerford
2026            Poshter Girl
8247    The Christmas Candle
2323          The Mirror Boy
3352         We Are the Wave
Name: title, dtype: object
```

ERROR & RESOLUTION LOG

It was discovered that there are problems with student accounts on Microsoft Azure. At the critical step selecting the node size to create the Kubernetes cluster, the system will not allow you to choose *any* size no matter the region, as shown below.





If you experience this problem, the only solution is to upgrade your account to the standard Pay As You Go account which will have the resources to obtain the proper resources. You may even have to create a brand new upgraded account to get it working.

GITHUB REPOSITORY

- https://github.com/CIS325-Group2/MovieRecommenderSystem

AUTHORS

- Nicholas Harrison
- Somi Poorey
- Kristy Seelnacht-Colombo

VERSION HISTORY

- 1.0 – Initial Release