ML Challenge – ACID LAB
Technical Manual

General Description

Project Source: https://github.com/jossteeven/acid-labs-devops-challenge

In this project you will find the following file structure:

- **.github/workflows/:** Github Actions Workflow pipelines
  - [main.yaml -> for the app implementation proccess]
  - [infrastructure_pipeline.yaml -> only for the infrastructure implementation, in this case Heroku]
- **app/:** Directory with all the files neccesary to build the REST API app. Python app and ML model can be found here.
- **datasets/:** Group of datasets to train and test the model in use.
- **infrastructure/:** Terraform files to handle the Heroku app implementation.
- **.gitignore:** Git .gitignore to exclude not needed temporal files.
- **Dockerfile:** Dockerfile needed to build the app image to deploy to Heroku app.
- **model.ipynb:** Model Jupyter Notebook

Used Tools

Heroku: Cloud used to create a web app instance to expose RESP API.
Docker: To test and build the image generated to run into the Heroku app instance.
Flask: Framework to load the .pkl model as a REST API.
Terraform: To handle the Heroku infrastructure implementation.
GitHub/GitHub Actions: To host the project and create CI/CD pipeline to automate the REST API deployment.

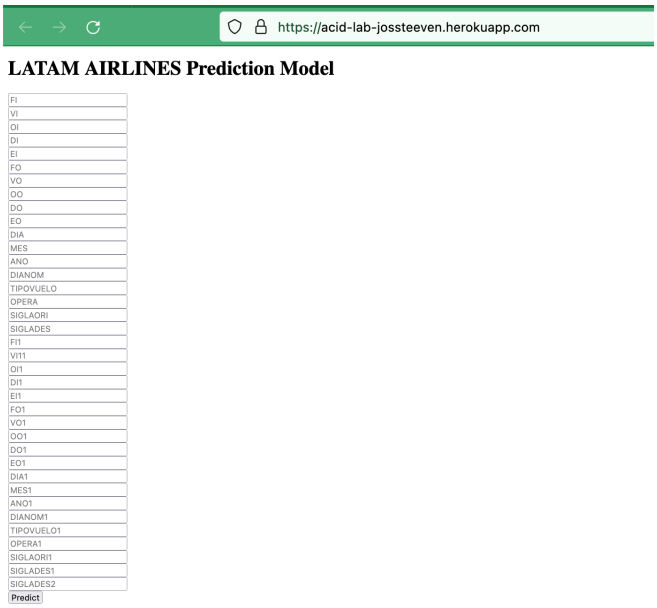
Project Implementation

REST API

To expose the pickle_model.pkl model as a REST API Flash was used as framework. This project can be found in the /app folder of this project.

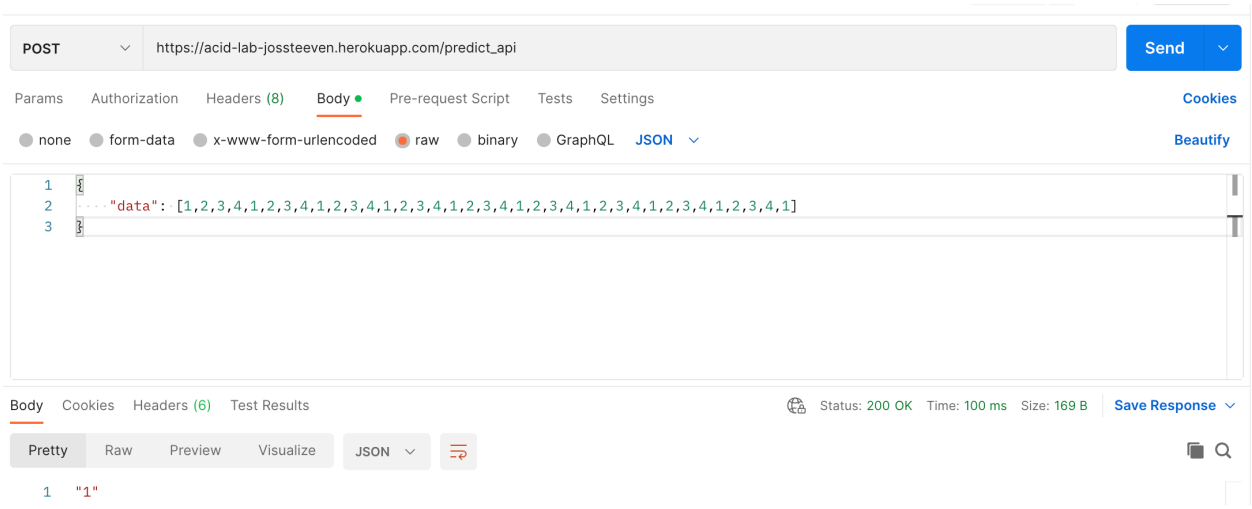
You can refer to the code annotation marked with ## to get a little more of detail of the code flow.

This REST API is expose in a Heroku web app as shown below:



This Web app redirects the form inputs to the route **/predict** that is seclusively used for the web UI to predict the flight delay.

For a quick test of the REST API model the **/predict_api** endpoint was expose so the usage of any CLI will be great to have the prediction values. An example usage of this endpoint is shown below.



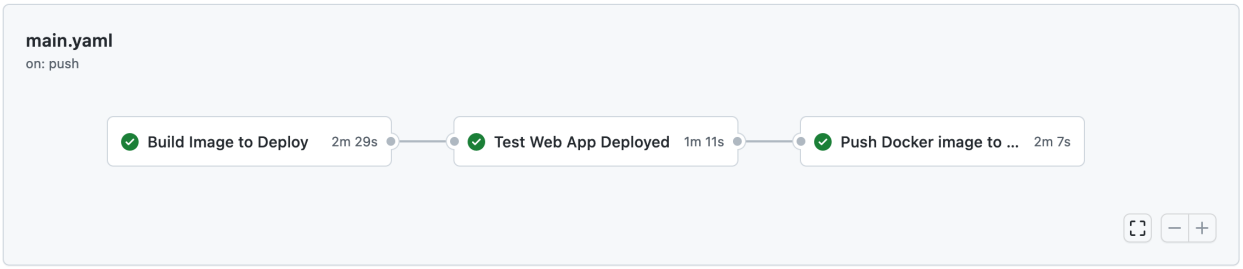
To pass the data array related to each of the 37 parameters to get a prediction you must pass as body a json as this:

```
{
  "data":[1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1]
}
```

Pipeline

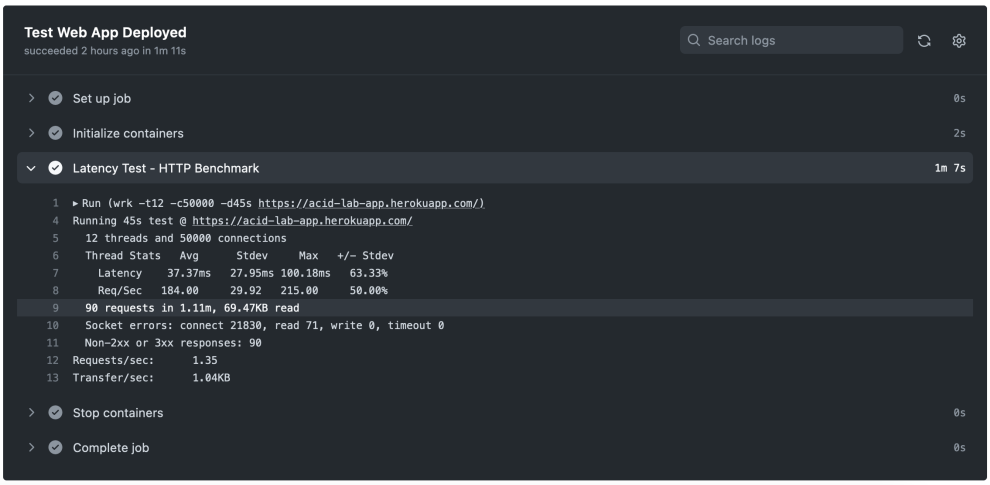
To be able to execute the automatic deployment form the project hosted in GitHub, a pipeline in GitHub Actions was built.

This pipeline has three stages as follows:

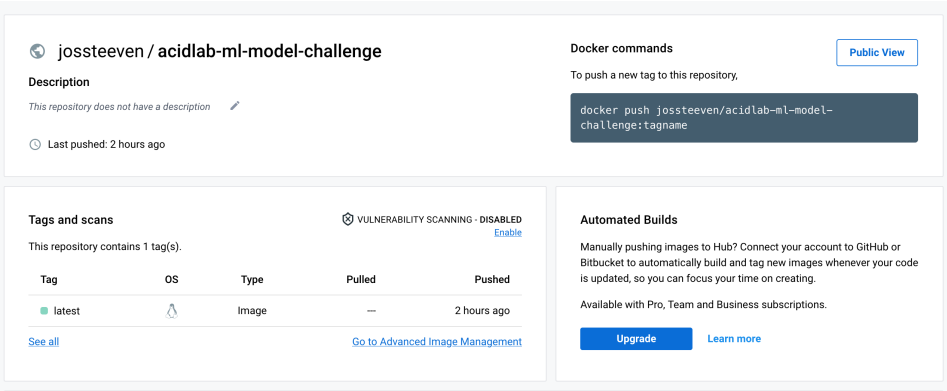


- **Build Image to Deploy**. - Stage responsible of generating the Docker image based on the Dockerfile you will find on the /app folder. This Dockerfile helps prepare the container environment with the necessary packages and resource to raise the REST API.

- **Test Web App Deployed**. – Stage responsible to generate the HTTP Benchmark as required to the challenge.



- **Push Docker Image to Heroku**. – Stage to generate and store the Docker image used to deploy to Heroku app. This Image can be pull from DockerHub as "*docker pull jossteeven/acidlab-ml-model-challenge:latest*"



Infrastructure

Terraform was used to raise the Keroku resources needed for these tests.
The following Terraform file's structure were implemented to accomplish the desired test.

- **main.tf:** Defines the resources to deploy. In our case defines the heroku_app resource.
- **variables.tf:** Defines the variables used in the Terraform configuration. The Heroku Api Key was defined in our case.
- **provider.tf:** Defines the Heroku provider configuration with user credentials.
- **versions.tf:** Defines the providers list. In our case we just define de Heroku provider with a specific version.

Be aware that for this project the infrastructure was handled locally because no backend status implementation was implemented for this case.
The status was saved in local files for easy purposes. TF_VAR_HEROKU_API_KEY was exported locally before running terraform init with the sensitive api key value.

HTTP Benchmark – WRK

As required. A way to implement the wrk test was to create a test job in the main pipeline of the project.
For this job the next script was used to execute it and get the report:

wrk -t12 -c50000 -d45s -s /data/request.lua https://acid-lab-jossteeven.herokuapp.com/predict_api

- **t12 : 12 threads**
- **c50000: 50000 requests**
- **d45: time of execution 45 seconds**
- **https://acid-lab-jossteeven.herokuapp.com/**

Additionally o custom wrk image was generated an stores in my personal docker regristry (DockerHub) **jossteeven/custom_wrk:latest.** The reason of generating this image was to use a request.lua file to use it during the call of the command to run the test.

This file request.lua has the following structure:

```
wrk.method = "POST"
wrk.body = "{\"data\": [1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1,2,3,4,1]}"
wrk.headers["Content-Type"] = "application/json"
```

This configuration allows us to change the requests configuration and include a body with this array to generate the 50000 requests for test.

The Dockerfile used to generate this image can we found at test/Dockerfile.

(Spanish) Extra Questions:

1. ¿Cómo podrías mejorar el performance de las pruebas anteriores?

Se podría controlar dinámicamente la ejecución de cada hilo asíncronamente y asociarlos en base a afinidad de cpu con el comando taskset y permitir así la ejecución paralela de miles de request.

Referencia: https://github.com/wg/wrk/issues/261#issuecomment-284027586

2. ¿Cuáles serían los mecanismos ideales para que sólo sistemas autorizados puedan acceder a esta API? (NO es necesario implementarlo).

Existen varios mecanismos que podrían limitar el acceso o el consumo de dicho API, entre ellos agregar un header de autenticación cuyo valor solo tengan las personas autorizadas y lo implementen al hacer llamados al API endopoint. La validación se realizaría en un API Proxy que podría estar ubicado en frente del API REST propio del modelo expuesto. Así también se podría enmascarar el endpoint de esta API REST desarrollada a través de API Proxies SaS como Apigeee, configurando un proxy con Access Control para Ips específicas y agregar ciertas validaciones adicionales como el header adicional ya mencionado o algún tipo de autenticación basado en user/password.

c. ¿Este mecanismo agrega algún nivel de latencia al consumidor? ¿Por qué?

El utilizar un Api Proxy por delante de una REST API agrega latencia de alguna forma, pudiendo llegar a afectar la calidad de experiencia de usuario. Esta latencia, aunque mínima, puede ser crítica para ciertas aplicaciones y es agregada porque la arquitectura presentaría ahora un nuevo filtro por el que los requests deben pasar. Para acceder al API REST (backend) deberán ahora pasar por las validaciones del API Proxy, agregando así latencia en el response.

d. ¿Cuáles serían los SLIs y SLOs que definirías y por qué?

Ya que los SLO se relacionan al equipo de trabajo, aquellos que plantearía son los siguientes:

- Mejorar el rendimiento de respuesta del modelo ante la cantidad de peticiones distribuidas globalmente al API REST.
- Disminuir el tiempo de indisponibilidad del API REST controlando los recursos de las instancias que conforman su arquitectura.
- Incrementar la eficiencia y precisión del modelo de predicción realizando nuevos entrenamientos y por ende nuevos despliegues de código.

Y los SLI asociados para cada SLO respectivamente serian:

- Response Time
- Web App Container Health Metric over time
- Parámetros de confianza del modelo predictive y de las predicciones que está arrojando en producción.
-

Definiría estas métricas ya que al tratarse de un modelo predictivo y que posiblemente sea usado en una página web distribuida y de consumo masivo, el mantener el API REST activo 24/7 y con una tolerancia a grandes cantidades de tráfico es indispensable por el tipo de plataformas que consumirían esta API. Mencionando ejemplos, estas plataformas podrían ser sistemas de tráfico aéreo comercial o relacionadas. Así también, considero importante que el mantener el modelo lo más preciso posible permitirá tener un sistema aún más exacto, por lo que su constante mejora es necesaria a través de nuevos entrenamientos y actualizaciones de código.