Nicole Gallo
8/25/2025
D602 - QBN1 Task 3: Program Deployment

COMPETENCIES
4162.1.3 : **Implements a Function**
The learner implements a function to call and receive information between multiple systems for deployment.
4162.1.4 : **Deploys a Data Based Product**
The learner deploys a data product based on project requirements.

In this task, you will write an API in Python in GitLab. **You will then create a Dockerfile that packages your API code and runs a web server to allow HTTP requests to your API. You will then explain how you wrote your code and the challenges you overcame. Finally, you will provide a video demonstrating the live API running from a deployed Docker container.**

A. *Create your subgroup and project in GitLab using the provided web link and the "GitLab How-To" web link by doing the following:*
    1. *Clone the project to the IDE.*
    2. *Commit with a message and push when you complete each requirement listed in parts B and D.*

       Note: You may commit and push whenever you want to back up your changes, even if a requirement is not yet complete.

    3. *Submit a copy of the GitLab repository URL in the "Comments to Evaluator" section when you submit this assessment.*
    4. *Submit a copy of the repository branch history retrieved from your repository, which must include the commit messages and dates.*

B. *Write an API with the code templates provided in either the FastAPI package in Python or the plumber package in R that accepts the following HTTP endpoints. Submit at least **two** versions of your code to the GitLab repository demonstrating a progression of work on your code.*
    1. *"/" should return a JSON message indicating that the API is functional.*
    2. *"/predict/delays" should accept a GET request specifying the arrival airport, the local departure time, and the local arrival time. It should return a JSON response indicating the average departure delay in minutes.*

C. *Write at least **three** unit tests for your API code, using the pytest package in Python or the testthat package in R, that test features of endpoints given both correctly formatted and incorrectly formatted requests. Submit at least **two** versions of your code to the GitLab repository demonstrating a progression of work on your code.*

D. *Write a Dockerfile referencing the requirements.txt file as appropriate that packages your API code and runs a web server to allow HTTP requests to your API. Submit at least **two** versions of your Dockerfile to the GitLab repository demonstrating a progression of work on your code.*

E. **Provide an explanation of how you wrote your code, including any challenges you encountered and how you addressed those challenges**.

I developed, tested, and containerized an API to serve predictions from my trained machine learning model. The following is an explanation of how I wrote the code and how I resolved any challenges I encountered during the process.

Writing the API Code
I started with the provided API_Python_1.0.0.py template, which already included the structure for loading the airport encodings file and a helper function for one-hot encoding arrival airports. To make the API function, I first added a FastAPI application instance and created a root endpoint (/) that returned a simple JSON message. This allowed me to confirm that the API could start and respond to requests.

Next, I implemented the /predict/delays endpoint. This endpoint accepts three inputs: the arrival airport code, the local departure time, and the local arrival time. Within the endpoint, I added logic to:

- Parse the departure and arrival times into seconds since midnight.
- Encode the arrival airport using the provided one-hot encoding helper function.
- Build the feature array in the exact order required by the model: (polynomial order, encoded airport array, departure time in seconds, arrival time in seconds).
- Load the trained model artifact (`finalized_model.pkl`) and generate a prediction of the average departure delay in minutes.
- Return the results in a JSON response.

Challenge: When I first attempted to start the API with `uvicorn`, I received an import error (`ModuleNotFoundError`) because the file name contained periods (`API_Python_1.0.1.py`).

Solution: I resolved this by renaming the file to `api_python.py`. This allowed `uvicorn` to properly import the module and start the server.

Writing the Unit Tests
To validate the API, I created automated tests using `pytest` and FastAPI's `TestClient`. I wrote three tests:

1. A test for the root endpoint (`/`) to confirm it returned a `200` status code and the expected JSON message.
2. A test for `/predict/delays` with valid inputs, which confirmed that the endpoint returned a `200` status code and included a numeric value for `average_departure_delay_min`.
3. A test for `/predict/delays` with invalid input, such as an incorrectly formatted time, which correctly returned a `422` error.

Challenges: Initially, running `pytest` failed because dependencies such as `httpx` and `certify` (used internally by FastAPI's `TestClient`) were missing. I also received a depreciation warning from `NumPy` about converting an array to a scalar.

Solutions: I updated my environment (switched to `conda`) by explicitly installing `httpx` and `certify`, which resolved the missing dependency errors. To address the `NumPy` warning, I modified the code to safely extract a scalar prediction value using `float(np.asarray(y).reshape(-1)[0])` instead of directly using `y[0]` (NumPy Manual).

Writing the Dockerfile
Once the API and tests were functioning, I wrote a `Dockerfile` to containerize the application. The `Dockerfile` used a lightweight Python base image, installed

dependencies from `requirements.txt`, copied the code and artifacts into the container, and started the API with `uvicorn` on `port 8000`.

Next, I created a non-root user for better security and included a health check that automatically pings the root endpoint (`/`) to confirm the container is healthy.
Challenge: The initial GitLab pipeline failed with the message "`No suitable configuration for the build context have been found.`"

Solution: I moved the `Dockerfile` to the repository root so the build context was correctly identified by GitLab's Kaniko runner.

F. Panopto recording:
NGallo_D602_Task3 Recording

G. Sources –
The only other external source used to complete this task beyond the sources provided in the coursework is the following:

1. NumPy Manual. *numpy.reshape.* https://numpy.org/doc/stable/reference/generated/numpy.reshape.html