

# D4.2 Prototype package (first release)

## Dispatcher3 – User manual

Deliverable 4.2

Dispatcher3

Grant:	886461
Call:	H2020-CS2-CFP10-2019-01
Topic:	JTI-CS2-2019-CfP10-SYS-01-16
Consortium coordinator:	University of Westminster
Dissemination level:	Confidential
Edition date:	31 October 2022
Edition:	01.01

## Authoring & Approval

### Authors of the document

Name/Beneficiary	Position/Title	Date
Luis Delgado / University of Westminster	Project member	27 July 2022
Clara Argerich Martín / Innaxis	Project member	27 July 2022
Ernesto Gregori / Innaxis	Project member	27 July 2022
Sergi Mas-Pujol/ Universitat Politècnica de Catalunya	Project member	27 July 2022

### Reviewers internal to the project

Name/Beneficiary	Position/Title	Date
Luis Delgado / University of Westminster	Project member	31 October 2022
Sergi Mas-Pujol/ Universitat Politècnica de Catalunya	Project member	31 October 2022

### Approved for submission to the CSJU

Beneficiary	Position	Date
University of Westminster	Project coordinator	31 October 2022

### Document History

Edition	Date	Status	Author	Justification
01.00	27 July 2022	Release	Dispatcher3 Consortium	New document for review by Topic Manager
01.01	31 October 2022	Release	Dispatcher3 Consortium	Document for review by CSJU

The opinions expressed herein reflect the authors' view only. Under no circumstances shall the Commission or Clean Sky Joint Undertaking be responsible for any use that may be made of the information contained herein.

# Dispatcher3

## INNOVATIVE PROCESSING FOR FLIGHT PRACTICES

This deliverable is part of a project that has received funding from the Clean Sky Joint Undertaking under grant agreement No 886461 under European Union's Horizon 2020 research and innovation programme.



### Abstract

---

This deliverable along with deliverable *D4.1. Technical documentation first release* consists of the release of the **first prototype of Dispatcher3**. The release consists of the binaries and Docker version of the prototype (sent to the Topic Manager).

The first release prototype package consists of a set on individual machine learning models which can be executed using Jupyter notebooks. It also includes the integration of the outcome of some of these individual models into a visualisation which would be part of the advice generator to provide high-level information to the end users. All models described in the Deliverable D4.1 will be available and executable in this release.

Data required to run the models (with some examples) are also provided. If data are public raw sample values are provided, otherwise pre-computed features are delivered so that the models can be run on individual flight examples. The prototypes can be run using local data (provided in the release) or with data stored in cloud storage (Amazon Web Services (AWS)).

This deliverable serves as a manual for the execution of the first release prototype software.

# Table of Contents

- Abstract.....3**
- 1 Introduction .....6
- 2 License .....8
- 3 Release components .....10
  - 3.1 Binaries .....10**
  - 3.2 Docker .....11**
- 4 Installation .....12
  - 4.1 Binaries installation.....12**
  - 4.2 Docker installation .....12**
- 5 Execution.....13
  - 5.1 Summary of executable notebooks .....13**
  - 5.2 Execution approaches .....17**
- 6 Git – Readme.....20
- 7 Acronyms .....21



## List of figures

Figure 1. CS100-SC001-E001-Fuel_model.ipynb example .....	17
Figure 2. Example docker execution .....	18
Figure 3. Notebooks available in Jupyter.....	19

## List of tables

Table 1. Release approaches .....	7
Table 2. Most significant files/packages included in binaries.....	10
Table 3. Notebooks and models from first release description.....	14

# 1 Introduction

---

Dispatcher3 is organised in three layers:

- **Data infrastructure:** Amazon Web Services (AWS), a multi-sided, data storage and processing platform. It provides private environments, secure dataframes, machine learning development frameworks and a scalable cloud computing and storage infrastructure.
- **Predictive capabilities,** provided by two distinct modules:
  - Data acquisition and preparation, with a first phase of data wrangling and a second step of descriptive analytics.
  - Predictive models, consisting on target variable labelling, feature engineering, training, testing, and validation of machine learning models.
- **Advice capabilities:** producing specific advice to targeted roles (dispatcher and pilots).

This first release contains individual trained machine learning models to predict different key indicators at two prediction horizons (3-4h prior SOBT and D-1). The integration of the outcome of some of these models into a coherent visualisation as a first step towards the advice capabilities is also presented.

More details on Dispatcher3 can be found in D1.1 - Technical Resources and Problem Definition - V.01.01. This release is accompanied of D4.1 - Technical documentation first release, which presents a high-level view of Dispatcher3 architecture and a detailed description of each of the models with information on data used, feature analysis and models performance. Further details on the model will be described on D4.3 - Technical documentation final release, which will include also information on the pipeline to prepare the datasets and train the models, and D5.2 - Verification and validation report with information on the models performance.

Jupyter notebooks are provided able to run each of the different trained machine learning models (<https://jupyter.org/> (Accessed July 2022)). They are accompanied by sample data input for the models. If data is public raw dataset are provided, otherwise, pre-computed features are released. In the final version the pipeline to generate these features will also be provided. For each notebook/model, the following steps are provided:

- Example data loaded (sample flight to which the prediction will be computed).
- Data preparation from example data for feature computation.
- Features computation (as they are the input into the machine learning models).
- Model loading and execution with features computed in previous step.

The release provides components that allow two approaches to install and execute the models as indicated in **Table 1**:

1. Python binaries: These components include the binaries of the software for a Linux environment. To execute the software a Linux host with specific libraries and packages is required.
2. Docker image: The docker image contains everything that is required to execute the software on any host (Linux, Windows, Mac) subject to the installation of the required Docker desktop software.

The Docker approach is more portable and recommended to avoid having to use a specific machine configuration.

**Table 1. Release approaches**

	Python binaries	Docker
Installation	Create Dispatcher3 environment installing python3 requirement	Loading Dispatcher3 docker into host machine
Installation requirements	Linux (5.4.0) Python3	Docker desktop installed in host machine (any operating system)
Release component	dispatcher3-binaries.zip	dispatcher3-docker.zip
Execution	Run Jupyter notebook to execute the notebooks with the models	Run docker this will automatically launch Jupyter notebook inside the docker

This document presents the release components, how to install and execute them, and briefly describes the files required as input and produced as outputs. The document is structured as follows

- Section 2- License and copyright notice.
- Section 3 - Release components (binaries, docker).
- Section 4 - Installation (using binaries or docker).
- Section 5 - Execution (including information on the different models provided).
- Section 6 - readme as in git repository.

## 2 License

---

Dispatcher3 prototype - D4.2 - Innovative Processing for Flight Practices software package (first release).

This software release is the confidential first release of Dispatcher3 prototype for sole use by Thales AVS, France (the licensee). Unauthorised redistribution of these files, via any medium, is strictly prohibited. The licensee may not reverse engineer, decompile, disassemble, or otherwise attempt to discover the source code of the software.

The software is provided 'as is' without warranty of any kind, either express or implied.

Authorship - Copyright Dispatcher3 components - packages (identified where file extensions are not given) and files:

- Copyright © 2022 Universitat Politècnica de Catalunya - All rights reserved:
  - `libs/atfm_scenarios/libs_airport_information.py`
  - `libs/atfm_scenarios/libs_metar_feature_extraction.py`
  - `libs/atfm_scenarios/libs_predictions_visualization.py`
  - `libs/turnaround_scenarios/libs_airport_information.py`
  - `libs/turnaround_scenarios/libs_metar_feature_extraction.py`
  - `libs/turnaround_scenarios/libs_operational_time.py`
  - `notebooks/CS500-SC005-E002-XXXX-XXXX_Turnaround_time.ipynb`
  - `notebooks/CS500-SC00X-E002-ATFM_delay.ipynb`
- Copyright © 2022 Fundación Instituto de Investigación Innaxis - All rights reserved:
  - `libs/fuel_scenarios/libs_fuel.py`
  - `libs/holding_scenarios/feature_extraction.py`
  - `libs/runway_scenarios/libs_runway.py`
  - `notebooks/CS100-SC001-E001-Fuel_model.ipynb`
  - `notebooks/CS100-SC004-E001-LEBL-EGLL_Holding_time.ipynb`
  - `notebooks/CS200-SC001-E001-XXXX-EGLL_Holding_time.ipynb`
  - `notebooks/CS200-SC001-E002-XXXX-EGLL_Holding_time.ipynb`
  - `notebooks/CS200-SC001-E003-XXXX-EGLL_Holding_time.ipynb`
  - `notebooks/CS300-SC001-E001-XXXX-LEIB_Runway.ipynb`



- Copyright © 2022 University of Westminster - All rights reserved:
  - libs/spark\_utilities.py
  - libs/utilities.py

## 3 Release components

---

This first release contains the binaries of Dispatcher3 compiled for Linux (5.4.0) and Dispatcher3 docker.

### 3.1 Binaries

The following table (**Table 2**) contains the most significant files included in dispatcher3-binaries.zip (prepared for execution in a Linux Ubuntu (5.4.0 - Ubuntu 20.04) environment).

**Table 2. Most significant files/packages included in binaries**

File	Note
README.md	Information about the repository.
license.txt	Copyright and authorship document.
advice_generator.templates	Advice Generator templates
input_data	Data required to run the different models with some examples.  Data is structured as follows: <ul style="list-style-type: none"> <li>• folder per model domain: <i>atfm_scenarios</i>, <i>fuel_scenarios</i>, <i>holding_scenarios</i>, <i>runway_scenarios</i>, <i>turnaround_scenarios</i></li> <li>• for each model domain:               <ul style="list-style-type: none"> <li>○ folder with information regarding each specific case study/experiment, e.g. <i>cs500_scXXX_e002</i></li> </ul> </li> </ul> <i>shared_input</i> with data which are shared for all experiments within the model domain
libs	Package with libraries used for the first release of Dispatcher3.  A library is defined per model domain: <i>atfm_scenarios</i> , <i>fuel_scenarios</i> , <i>holding_scenarios</i> , <i>runway_scenarios</i> , <i>turnaround_scenarios</i>  These contain the compiled libraries required to compute the features used by the machine learning models.

File	Note
ml_models	Package which includes the trained machine learning models. There is one sub-package per model, e.g. <i>fuel_cs100_sc002_e001</i> .
notebooks	Package with all the models executable notebooks: <ul style="list-style-type: none"> <li>• <i>CS100-SC001-E001-Fuel_model.ipynb</i></li> <li>• <i>CS100-SC004-E001-LEBL-EGLL_Holding_time.ipynb</i></li> <li>• <i>CS200-SC001-E001-XXXX-EGLL_Holding_time.ipynb</i></li> <li>• <i>CS200-SC001-E002-XXXX-EGLL_Holding_time.ipynb</i></li> <li>• <i>CS200-SC001-E003-XXXX-EGLL_Holding_time.ipynb</i></li> <li>• <i>CS300-SC001-E001-XXXX-LEIB_Runway.ipynb</i></li> <li>• <i>CS500-SC00X-E002-ATFM_delay.ipynb</i></li> <li>• <i>CS500-SC005-E002-XXXX-XXXX_Turnaround_time.ipynb</i></li> </ul>

## 3.2 Docker

dispatcher3-docker.zip contains dispatcher3-docker.tar which is a docker image of Dispatcher3. More information about Docker can be found here: <https://www.docker.com/> (Accessed July 2022).

## 4 Installation

---

### 4.1 Binaries installation

A Python3 virtual environment is recommended even if not required. In the instructions below *dispatcher3-release* is used as environment name. If virtual environment is not used and AWS data is not used only step 1 and 3 are required:

1. Extract the content of dispatcher3-binaries.zip
2. Create virtual environment (not required if no virtual environment used):  
`mkvirtualenv dispatcher3-release`
3. Install Python3 requirements:  
`pip3 install -r requirements.txt`
4. If data to be accessed from AWS install required libraries (if already not available) (not required if only data in local (provided in the release) is to be used):  
`sudo apt-get install awscli`
5. If virtual environment is used, add the virtual environment to Jupyter notebook kernels:  
`ipython kernel install --name dispatcher3-release --user`

Note that if AWS is used, the AWS profile dispatcher3 with the required credentials should be defined beforehand. More information in: <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html> (Accessed July 2022). This is not required if the data provided in the release is directly used.

As explained in D4.1 - Technical documentation first release, the machine learning models have been trained using Databricks, therefore they can also be run within that infrastructure by loading the binaries into the platform. In this case if data is to be accessed from AWS then Databricks File System (DBFS) is used to link the AWS data lake with Databricks. Public and private AWS keys are used to mount a S3 (AWS) bucket into Databricks with the *mount* command (as described in <https://docs.databricks.com/data/data-sources/aws/amazon-s3.html#mount-aws-s3> (Accessed July 2022)). Note this is not required to run the models just if Databricks is used for execution.

### 4.2 Docker installation

1. Install Docker desktop in host machine if not already available. More information in <https://www.docker.com/get-started> (Accessed July 2022).
2. Unzip dispatcher3.zip to obtain dispatcher3-docker.tar
3. Load Dispatcher3 image:  
`docker load -i dispatcher3-docker.tar`

## 5 Execution

---

### 5.1 Summary of executable notebooks

**Table 3** contains a summary of the executable notebooks delivered in the first prototype release. The table collects the name of the notebook, the data sources required for executing it, and a list of the features that have been pre-computed (to alleviate computational time or to respect the data protection agreement). The sample flight(s) on which the model is(are) executed are also described with the output the notebooks.



DISPATCHER3

Table 3. Notebooks and models from first release description

Jupyter Notebook	Target output variable (Case study id (as in D4.1))	Data provided in release	Pre-computed features <sup>1</sup>	Flight example provided
CS100-SC001-E001-Fuel_model.ipynb	Fuel usage estimation for routes from LEBL to EGLL or EGKK (CS400-SC002-E001 - 4-LEBL-EGLL/EGKK)	METAR file NOAA file		<ul style="list-style-type: none"> <li>VLG7830 - LEBL – EGKK - 7/8/2018</li> </ul>
CS100-SC004-E001-LEBL-EGLL_Holding_time.ipynb	Holding time at arrival at EGLL in minutes (CS100-SC004-E001 - 4-LEBL-EGLL)	METAR file EUROCONTROL R&D Archive sample	Occupancy at departure time Occupancy at arrival time	<ul style="list-style-type: none"> <li>BAW475 - LEBL – EGLL - 12/09/2018</li> </ul>
CS200-SC001-E001-XXXX-EGLL_Holding_time.ipynb	Holding time at arrival at EGLL in minutes (CS200-SC001-E001- 4-xxx-EGLL)	METAR file EUROCONTROL R&D Archive sample	Occupancy at departure time Occupancy at arrival time	<ul style="list-style-type: none"> <li>AAL732 - KCLT – EGLL - 12/09/2018</li> </ul>
CS200-SC001-E002-XXXX-EGLL_Holding_time.ipynb	Probability of holding at arrival at EGLL and time of holding in minutes (CS200-SC001-E002- 4-xxx-EGLL)	METAR file EUROCONTROL R&D Archive sample	Occupancy at departure time Occupancy at arrival time	<ul style="list-style-type: none"> <li>SHT3U - EGCC – EGLL - 12/09/2018</li> </ul>
CS200-SC001-E003-XXXX-EGLL_Holding_time.ipynb	Holding severity at arrival at EGLL (negligible, soft, mild or severe) (CS200-SC001-E003- 4-xxx-EGLL)	METAR file EUROCONTROL R&D Archive sample	Occupancy at departure time Occupancy at arrival time	<ul style="list-style-type: none"> <li>SHT3U - EGCC – EGLL - 12/09/2018</li> </ul>



Jupyter Notebook	Target output variable (Case study id (as in D4.1))	Data provided in release	Pre-computed features <sup>1</sup>	Flight example provided
CS300-SC001-E001-XXXX-LEIB _Runway.ipynb	Runway in use at LEIB (CS300-SC001-E001 - 4-xxx-LEIB)	METAR file  EUROCONTROL R&D Archive		<ul style="list-style-type: none"> <li>VLG3526 - LEBL – LEIB - 4/09/2018</li> </ul>
CS500-SC005-E002-XXXX-XXXX _Turnaround_time.ipynb	Turnaround time for flights which have been issued an ATFM delay (CS500-SC005-E002-24-xxx-xxx)	METAR file  Airlines static information  Airports static information	Occupancy airport at departure time  Occupancy airport at arrival time	<ul style="list-style-type: none"> <li>LEBL – LLBG - 10/08/2018</li> <li>LCLK – LEBL - 10/08/2018</li> <li>LFPO – LPPT - 10/08/2018</li> </ul>

Jupyter Notebook	Target output variable (Case study id (as in D4.1))	Data provided in release	Pre-computed features <sup>1</sup>	Flight example provided
CS500-SC00X-E002-ATFM_delay.ipynb	<ul style="list-style-type: none"> <li>Probability of ATFM delay (CS500-SC001-E002-24-xxx-xxx)</li> <li>Location of ATFM regulation for regulated flights (airspace vs aerodrome) (CS500-SC002-E002-24-xxx-xxx)</li> <li>Non-zero probability of ATFM delay for regulated flights (CS500-SC003-E002-24-xxx-xxx)</li> <li>ATFM delay and uncertainty in minutes for regulated flights (CS500-SC004-E002-24-xxx-xxx)</li> </ul> <p>Note that this notebook provides an integration of the outcome of the four models into a preliminary view of the Advice Generator for ATFM delay.</p>	<ul style="list-style-type: none"> <li>METAR file</li> <li>Airlines static information</li> <li>Airports static information</li> </ul>	<ul style="list-style-type: none"> <li>Occupancy airport at departure time</li> <li>Occupancy airport at arrival time</li> <li>Demand airspace</li> </ul>	<ul style="list-style-type: none"> <li>VLG2021 - LEGR – LEBL - 07/08/2018</li> <li>VLG3314 - LIRF – LGSR - 07/08/2018</li> <li>VLG51XP - EDDM – LEBL - 07/08/2018</li> <li>VLG35UU - LEIB – LEBL - 07/08/2018</li> <li>VLG3303 - LIRF – LEIB - 07/08/2018</li> </ul>

<sup>1</sup> As computed from EUROCONTROL DDR2 repository.



## 5.2 Execution approaches

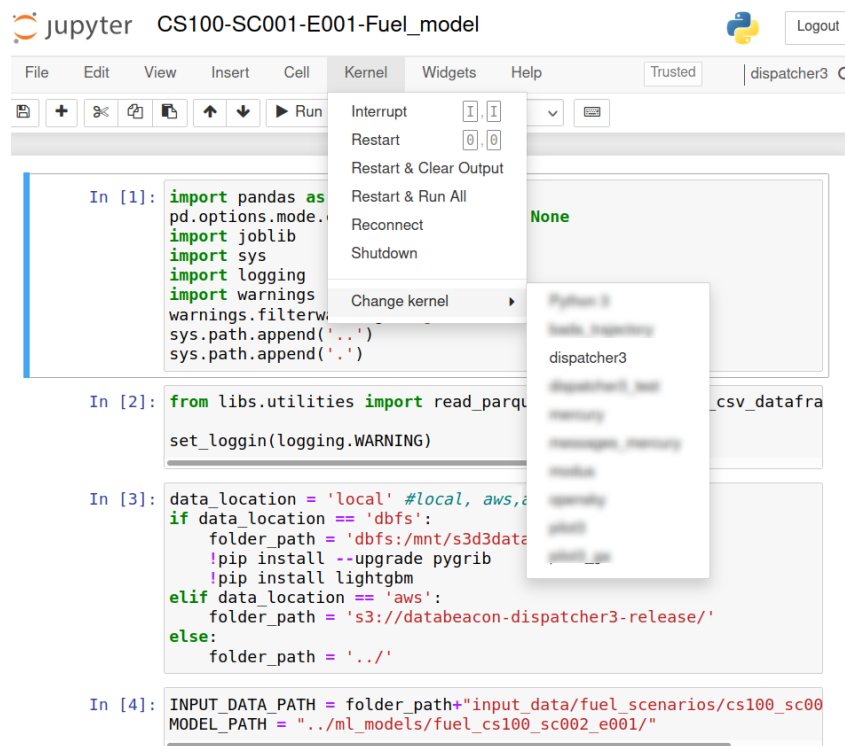
### 5.2.1 Binaries execution

#### 5.2.1.1 Nominal execution

Once the release has been installed. The steps required to run Jupyter notebook within the virtual environment:

```
$(pwd)/dispatcher3$ jupyter notebook
```

This will open Jupyter in a browser. Enter in notebooks folder and open the notebook which is going to be executed, e.g. *CS100-SC001-E001-Fuel\_model.ipynb*. Change the kernel to the one in the virtual environment in which the libraries have been installed (in the example in **Figure 1** *dispatcher3* virtual environment is used).



The screenshot shows a Jupyter Notebook titled "CS100-SC001-E001-Fuel\_model" with a Python kernel. The interface includes a menu bar (File, Edit, View, Insert, Cell, Kernel, Widgets, Help) and a toolbar with icons for running and interrupting. A code cell is selected, and the "Kernel" menu is open, showing options like "Interrupt", "Restart", "Restart & Clear Output", "Restart & Run All", "Reconnect", "Shutdown", and "Change kernel". The "Change kernel" submenu is also visible, listing available kernels: "Python 3", "dispatcher3", and "dispatcher3".

```
In [1]: import pandas as pd
pd.options.mode.chained_assignment = None
import joblib
import sys
import logging
import warnings
warnings.filterwarnings('ignore')
sys.path.append('.')
sys.path.append('.')

In [2]: from libs.utilities import read_parquet
set_loggin(logging.WARNING)

In [3]: data_location = 'local' #local, aws, dbfs
if data_location == 'dbfs':
    folder_path = 'dbfs:/mnt/s3d3data'
    !pip install --upgrade pygrib
    !pip install lightgbm
elif data_location == 'aws':
    folder_path = 's3://databeacon-dispatcher3-release/'
else:
    folder_path = './'

In [4]: INPUT_DATA_PATH = folder_path+"input_data/fuel_scenarios/cs100_sc001/"
MODEL_PATH = "../ml_models/fuel_cs100_sc002_e001/"
```

**Figure 1.** CS100-SC001-E001-Fuel\_model.ipynb example

All notebooks contain an explanation of the different processes from loading the data (with the flight example), loading the model, computing the features required as input for the models and running the models. Use the *Run* command for each of the cells in the notebook. See **Figure 3** for a list of the notebooks in Jupyter.

#### 5.2.1.2 Data in AWS

By default *data\_location* (cell 3 in **Figure 1**) should be defined as 'local' to use the data provided in the release. This variable can be changed to 'aws' if data stored in AWS *databeacon-dispatcher3-release* bucket is used, or 'dbfs' if the notebooks are executed inside Databricks.

Note that if AWS is used AWS\_PROFILE system variable needs to be defined in the system prior executing Jupyter notebook. Therefore, before executing 'jupyter notebook' set the system variable with the command 'export AWS\_PROFILE=dispatcher3' in the same terminal.

The AWS profile dispatcher3 with the required credentials should be defined beforehand. More information in <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html> (Accessed July 2022).

This is a template of the AWS credentials profile file which should be stored in ~/.aws/credentials:

```
[dispatcher3]
aws_access_key_id =
aws_secret_access_key =
region=eu-west-1
```

## 5.2.2 Docker execution

### 5.2.2.1 Nominal execution

Once the docker image has been installed in the system to launch the docker execute:

```
docker run --rm --name jupyter_dispatcher3 -p 8888:8888 dispatcher3
```

Make sure that port 8888 is not being used by another programme in the system (e.g. another Jupyter notebook). Otherwise an 'Error starting userland proxy: bin: address already in use' error will be generated, a different port could be used in that case (e.g. -p 8889:8888 to use port 8889 in the host machine instead of 8888).

In the console the docker will print the URL and token required to launch the notebook in a browser to connect to the docker, as shown in **Figure 2**. In this case the url is: <http://127.0.0.1:8888/?token=5d62ce398029a41a96da13a9f26d266311a596d784742b76>. Instead of the URL with the token, the user can also open URL <https://127.0.0.1:8888> which will then request the token to access the notebooks.

```

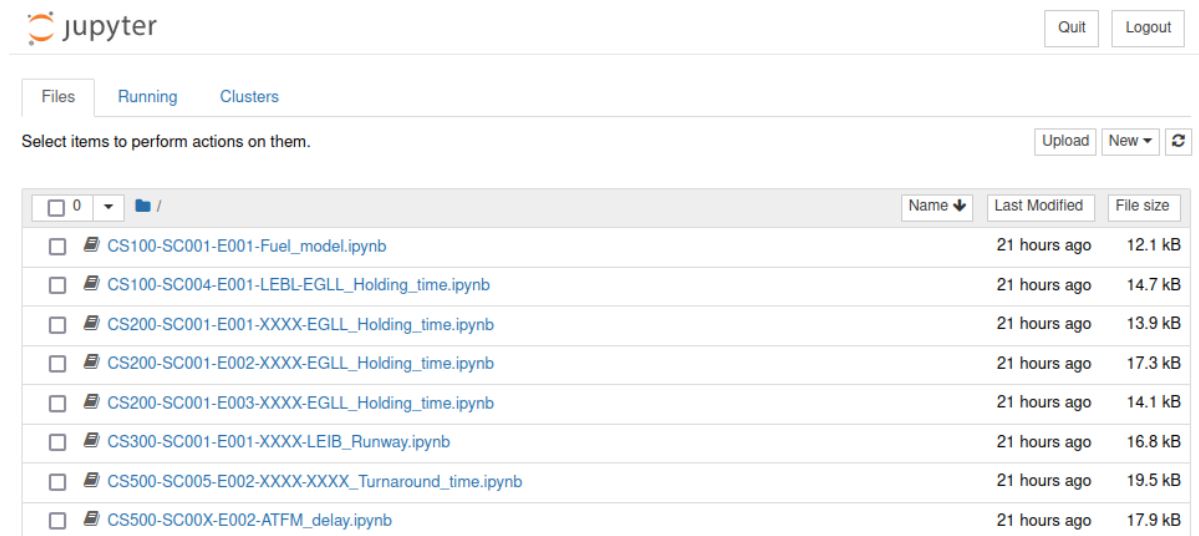
Luis@jura:~/projects/dispatcher3$ docker run --rm --name jupyter_dispatcher3 -p 8888:8888 dispatcher3
[I 11:30:55.225 NotebookApp] Writing notebook server cookie secret to /root/.local/share/jupyter/runtime/notebook_cookie_secret
[I 11:30:55.450 NotebookApp] Serving notebooks from local directory: /src/dispatcher3/notebooks
[I 11:30:55.450 NotebookApp] Jupyter Notebook 6.3.0 is running at:
[I 11:30:55.450 NotebookApp] http://a99a44645977:8888/?token=5d62ce398029a41a96da13a9f26d266311a596d784742b76
[I 11:30:55.450 NotebookApp] or http://127.0.0.1:8888/?token=5d62ce398029a41a96da13a9f26d266311a596d784742b76
[I 11:30:55.450 NotebookApp] Use Control-C to stop this server and shut down all kernels (twice to skip confirmation).
[C 11:30:55.454 NotebookApp]

To access the notebook, open this file in a browser:
file:///root/.local/share/jupyter/runtime/nbserver-6-open.html
Or copy and paste one of these URLs:
http://a99a44645977:8888/?token=5d62ce398029a41a96da13a9f26d266311a596d784742b76
or http://127.0.0.1:8888/?token=5d62ce398029a41a96da13a9f26d266311a596d784742b76

```

**Figure 2. Example docker execution**

Opening the URL in a browser will present the different notebooks available within the docker as shown in **Figure 3**.



The screenshot shows the Jupyter web interface. At the top, there are 'Quit' and 'Logout' buttons. Below that, there are tabs for 'Files', 'Running', and 'Clusters'. A message says 'Select items to perform actions on them.' with 'Upload', 'New', and a refresh icon. The main area is a table of files:

<input type="checkbox"/>	Name	Last Modified	File size
<input type="checkbox"/>	CS100-SC001-E001-Fuel_model.ipynb	21 hours ago	12.1 kB
<input type="checkbox"/>	CS100-SC004-E001-LEBL-EGLL_Holding_time.ipynb	21 hours ago	14.7 kB
<input type="checkbox"/>	CS200-SC001-E001-XXXX-EGLL_Holding_time.ipynb	21 hours ago	13.9 kB
<input type="checkbox"/>	CS200-SC001-E002-XXXX-EGLL_Holding_time.ipynb	21 hours ago	17.3 kB
<input type="checkbox"/>	CS200-SC001-E003-XXXX-EGLL_Holding_time.ipynb	21 hours ago	14.1 kB
<input type="checkbox"/>	CS300-SC001-E001-XXXX-LEIB_Runway.ipynb	21 hours ago	16.8 kB
<input type="checkbox"/>	CS500-SC005-E002-XXXX-XXXX_Turnaround_time.ipynb	21 hours ago	19.5 kB
<input type="checkbox"/>	CS500-SC00X-E002-ATFM_delay.ipynb	21 hours ago	17.9 kB

**Figure 3. Notebooks available in Jupyter**

Follow the same steps as with the binary execution version to run the models: open the notebook and run the different cells. By default *data\_location* should be defined as 'local' to use the data provided in the release which is already embedded in the docker.

To stop the docker run in a separate terminal: `docker stop jupyter_dispatcher3`

### 5.2.2.2 Data in AWS

As with the binary version, it is possible to use the data stored in AWS (*data beacon-dispatcher3-release*). In this case the credentials need to be set inside the Docker before executing Jupyter notebook. The steps to achieve this are:

1. Run the docker overriding the entry point so that instead of automatically executing Jupyter notebook the console inside the Docker is executed:  
`docker run --entrypoint "/bin/bash" -it --rm --name jupyter_dispatcher3 -p 8888:8888 dispatcher3`
2. Edit the `aws/credentials` file with the credentials to access the AWS bucket:  
`vi ~/.aws/credentials` [As the template provided above (see Section 5.2.1.2), add the credentials and save the file (wq!), more information in <https://docs.aws.amazon.com/cli/latest/userguide/cli-configure-profiles.html> (Accessed July 2022)]
3. Export the `AWS_PROFILE` system variable:  
`export AWS_PROFILE=dispatcher3`
4. Run jupyter notebook:  
`jupyter notebook --ip=0.0.0.0 --port=8888 --no-browser --allow-root --notebook-dir=/src/dispatcher3/notebooks/`

Follow the same steps as with the nominal execution (copy address in browser).

## 6 Git – Readme

---

Libraries and Notebooks for Dispatcher3 release.

It contains a set of notebooks to execute the trained machine learning models for the following cases:

- Holdings at arrival at EGLL
- Fuel usage in route from LEBL to EGLL/EGKK
- Runway at arrival at LEIB
- ATFM probability, location, non-zero and amount of delay
- Turnaround time if flight has ATFM delay assigned

The code is complemented with D4.1 - Technical documentation first release where detailed information on Dispatcher3 and the models (training, validation, results) are presented, and D4.2 - Prototype package (first release) - User manual, which explains how to use the code provided in this release.

## 7 Acronyms

---

ATFM: Air traffic flow management

AWS: Amazon Web Services

CSJU: Clean Sky 2 Joint Undertaking

DDR2: Demand data repository (second phase)

EDDM: Munich airport

EGKK: London Gatwick airport

EGLL: London Heathrow airport

H2020: Horizon 2020 research programme

KCLT: Charlotte airport

LCLK: Larnaca airport

LEBL: Barcelona airport

LEGR: Granada Jaen airport

LEIB: Ibiza airport

LFPO: Paris Orly airport

LGSR: Santorini airport

LIRF: Rome Fiumicino airport

LPPT: Lisbon airport

METAR: Meteorological aerodrome report

NOAA: National oceanic and atmospheric administration

R&D: EUROCONTROL research & development traffic data set

SOBT: Scheduled off-block time

-END OF DOCUMENT-

