Authors: **Oxoli Daniele** (daniele.oxoli@polimi.it), **Emanuele Capizzi** (emanuele.capizzi@polimi.it) - 2022 - Politecnico di Milano, Italy

WEkEO2Pydash - Explore Copernicus data interactively using the WEkEO HDA API

WEkEO Jupyter Notebook competition: https://notebook.wekeo.eu (**Track A**: *Exploit the broad range of Copernicus Data*)

NOTEBOOK INTRODUCTION

Outline

This Notebook showcases Python recipes to interact (access, browse, display and download) with the Copernicus data dispatched by the WEkEO DIAS, through the development of flexible and interactive dashboards into a Jupyter notebook.

Interactivity is here used as the key element to speed-up applications development by minimizing code editing for recursive steps such as variables definition and parameters setting.

The final goal is to provide the user with reusable code blocks which can be adapted - *with a small effort* - to manifold Copernicus data applications by leveraging the WEkEO Harmonised Data Access (HDA) API as exclusive data endpoint.

Resources

This Notebook makes extensive use of the WEkEO HDA API to perform `GET` and `POST` requests[1], necessary for automating the data access procedures.

Interactivity is enabled by cutting-edge Python libraries for dynamic widgets and maps generation including IPython, itables, ipywidgets and ipyleaflet; alongside popular data managing and analysis libraries such as Pandas and xarray. All the selected libraries are released under open-license[2] compatible with MIT license.

The pattern proposed by this Notebook is developed and demonstrated through examples, adapted to different data products[3] provided by the WEkEO DIAS. Specifically, the data products considered in this Notebook are reported in the following table:

Product Description	Product Link	ID	Metadata
ERA5 - Single Levels	link	EO:ECMWF:DAT:REANALYSIS_ERA5_SINGLE_LEVELS	link
CAMS - European Air Quality Forecasts	link	EO:ECMWF:DAT:CAMS_EUROPE_AIR_QUALITY_FORECASTS	link
Sentinel-5P	link	'EO:ESA:DAT:SENTINEL-5P:TROPOMI'	link

Settings to adapt the Notebook functions and dynamic widgets to the different data products are explained throughout the Notebook sections.

Learning outcomes

At the end of this Notebook you will know:

- How to programmatically access Copernicus data and metadata using the WEkEO HDA API in Python
- How to generate dynamic data previews using interactive Python widgets
- How to adapt and reuse Python functions and code blocks to deal with different WEkEO data products and applications

[[1]Swagger UI](https://wekeo-broker.apps.mercator.dpi.wekeo.eu/databroker/ui/#!/HDA_-_dataorder/dataorder_get)

[[2]About Open Source Licenses](https://opensource.org/licenses)

[[3]WEkEO Data Discovery Platform](https://www.wekeo.eu/data)

Content

➤ 0. From WEkEO Data Discovery Platform to Jupyter Notebook [Back to top](#TOC_TOP)

All WEkEO data can be manually downloaded from the WEkEO Data Discovery Platform. Before running the Notebook, you are requested to **create a personal account** on the WEkEO website.

Once registered, you can browse and select the datasets of interest directly from the WEkEO Data Discovery Platform GUI (see figure below) and proceed with the manual download.

- `A` = `Layers` functionality allows you to accessAfter getting confident with the WEkEO Data Discovery Platform GUI, you are ready to start programming for bringing your entire data access and analysis workflows into a Jupyter Notebook. the WEkEO Catalog and select the desired dataset. Using the `Add to map` button your dataset will be available for requesting the data.
- `B` = `Subset and download` button allows to select the desired values for each parameter and build the associated query.
- `C` = `Show layer information` shows all the information and metadata related to the specific dataset.
- `D` = `Jobs` functionality collects all the data requested previously. It allows to order the data and download them.

After getting confident with the WEkEO Data Discovery Platform GUI, you are ready to start programming for **bringing your entire data access and analysis workflows into a** Jupyter Notebook.

✔ Using the **HDA API** and Python, all the procedures for requesting the data can be replicated and automated in a programmatic way. Moreover, you will have the possibility of browsing, downloading, displaying and analysing data without leaving this Jupyter Notebook application!

⚠ The Notebook is meant to be used in **JupyterLab**. Some interactive widgets and functions may not work correctly if using it in a standard Jupyter Notebook application.

➤ 1. Python Environment and Libraries [Back to top](#TOC_TOP)

Before starting running the code, you have to set up a virtual Python environment and install all the requested libraries listed in the environment.yml file provided inside the Notebook repository.

You can find additional info on the WEkEO Storage and Python environments web page.

In [1]:
```
#Disable some Python warnings that may arise during the import
import warnings
warnings.filterwarnings('ignore')
warnings.simplefilter(action='ignore', category=UserWarning)

# Base libraries
import requests
import json
import zipfile
import os
import pandas as pd
from pandas.io.json import json_normalize
import base64
import datetime
import ipywidgets as widgets
from ipywidgets import Layout
import numpy as np
import xarray as xr
import rioxarray as rxr

#Utility libraries
from PIL import Image
from ipyleaflet import Rectangle
import IPython
from IPython.display import display, JSON, Image
from urllib.request import urlopen
import matplotlib.pyplot as plt
import matplotlib.ticker as mticker
from mpl_toolkits.basemap import Basemap
import cartopy.crs as ccrs
import cartopy.feature as cfeature
from cartopy.mpl.gridliner import LONGITUDE_FORMATTER, LATITUDE_FORMATTER
from netCDF4 import Dataset
from itables import init_notebook_mode, show
import itables.options as opt
opt.classes = ["display","hover", "nowrap"]
init_notebook_mode(all_interactive=True)
```

This Notebook makes use of custom Python functions which are stored in the file wekeo2pydash_methods.py.

In [2]:
```
#Import custom functions
import wekeo2pydash_methods as m
```

⚠️ Keep the **wekeo2pydash_methods.py** file in the same folder of the Notebook to import its functions into your workspace!

✔ If no errors appeared during the libraries import, you are ready to run the Notebook!

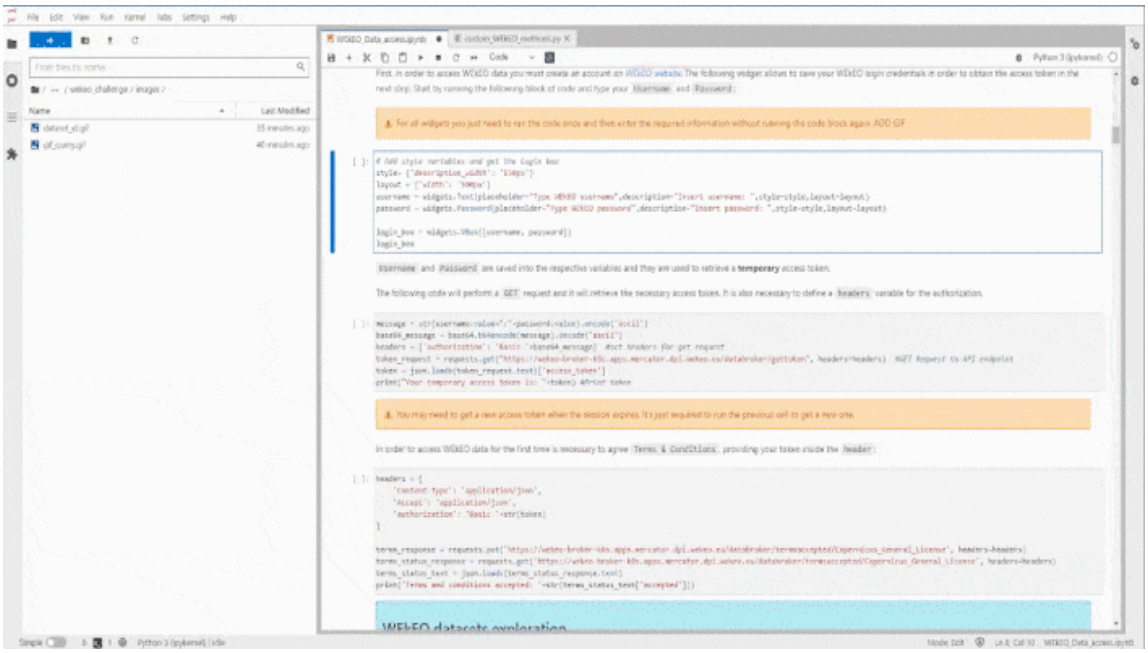
➤ 2. Login to WEkEO DIAS [Back to top](#TOC_TOP)

Once you registered to WEkEO website, you can run the following code that will generate a

widget which will allow you to insert your WEkEO `Username` and `Password` and obtain your temporary access `Token` (**valid for 1 hour**).

You can find additional info on the HDA API documentation - Authentication web page.

> ⚠️ For all the widgets, you just need to run the code once and then enter the required information inside the widget without running the code block again. Then, you can just go to the next one.



In [ ]:
```
# Add style variables and get the login widget
style = {'description_width': '150px'}  #styling
layout = {'width': '500px'}  #Layout
username = widgets.Text(placeholder = "Type here your WEkEO username", description
password = widgets.Password(placeholder = "Type here your WEkEO password", descrip

login_box = widgets.VBox([username, password])  #create the login boxes
login_box  #show login widget
```

`Username` and `Password` are saved into *Python variables (strings)* which are used in the next code block to retrieve the temporary access `Token`.

The access token has to be requested using the **HDA API**. The following code block will perform a `GET` request for such a purpose. The `headers` variable is required for the authorization.

In [5]:
```
message = str(username.value + ":" + password.value).encode('ascii')
base64_message = base64.b64encode(message).decode('ascii')
headers = {'authorization': 'Basic ' + base64_message}  #set headers for get reques
token_request = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/
```

```
token = json.loads(token_request.text)['access_token']  #save token
print("Your temporary access token is: " + token) #print token
```

```
Your temporary access token is: 10588914-3f1d-3c05-9feb-15cd37cddeb9
```

⚠️ You need to request a new Token if the session expires, otherwise, you will be not able to request any data. To do so, just run again the previous code block.

In order to access WEkEO data for the first time, it is necessary to accept the **Copernicus Terms & Conditions** by providing your token inside the `authorization` key contained in the `header`:

In [6]:
```
headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
    'authorization': 'Basic ' + str(token)
}

terms_response = requests.put('https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/
terms_status_response = requests.get('https://wekeo-broker-k8s.apps.mercator.dpi.we
terms_status_text = json.loads(terms_status_response.text)
print('Terms and conditions accepted: ' + str(terms_status_text['accepted']))
```

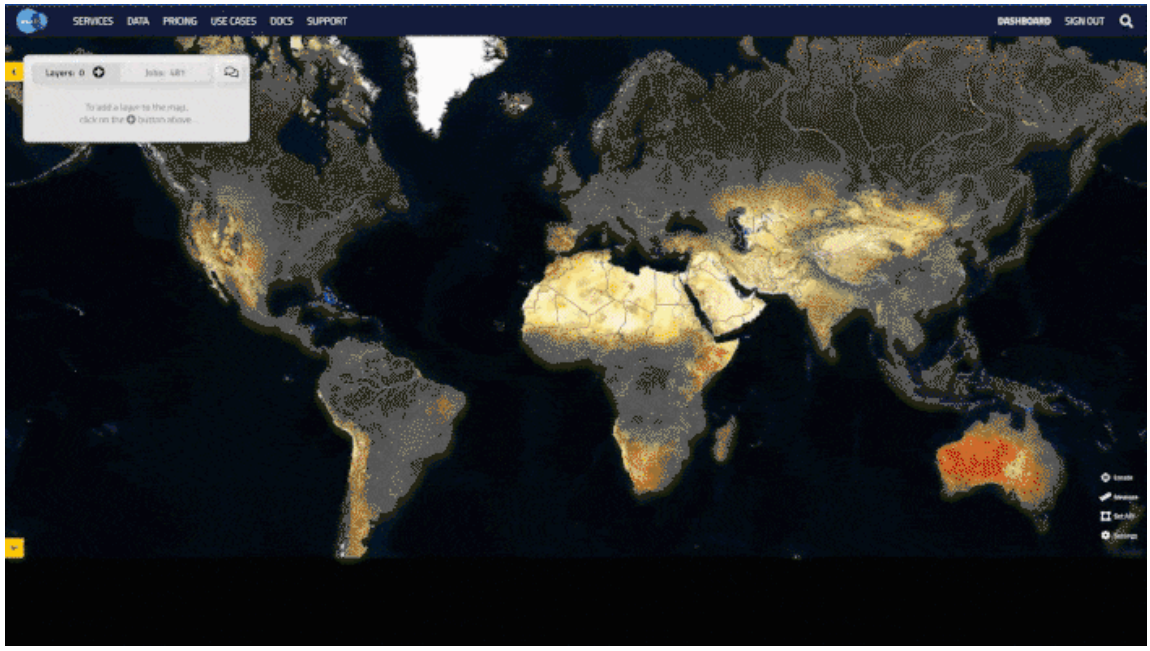
```
Terms and conditions accepted: True
```

✔ Now that you have obtained your Token, you are ready to discover the WEkEO data catalogue!

➤ 3. Browsing the WEkEO Data Catalogue [Back to top](#TOC_TOP)

To access a dataset, you have to look for the `datasetId` which is an identifier for a specific resource in the WEkEO data catalogue.

You can try manually to retrieve it from the WEkEO Data Discovery Platform.



An alternative is to access the catalogue by querying the HDA API using Python directly in the Notebook.

It is possible to enhance the exploration of the query result using by casting the catalogue list into a Pandas DataFrame. The itables library further allows adding interactivity to DataFrame exploration.

After running the following code block, you will be able to explore the catalogue, filter the records using keywords, search for a specific dataset, and retrieve its `datasetId`. You can decrease or extend the number of records in the query result by modifying the variable `size`.

In this example, only the `datasetId` and the `title` columns are kept in the table. However, additional information is available, as will be shown in Section 4.

In [ ]:
```
size = 2000 # Max number of datasets to be requested
dataset = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/databro
data = json.loads(dataset.text)
data_df = pd.json_normalize(data['content'])
data_show = data_df[['title','datasetId']]
show(data_show, scrollX=True, columnDefs=[{"classNames": "dt-left", "targets": "_all
```

Show 10 entries Search:

title	datasetId
10-daily Dry Matter Productivity 1KM	EO:CLMS:DAT:CGLS_GLOBAL_DMP_V2_1KM
10-daily Dry Matter Productivity 333M	EO:CLMS:DAT:CGLS_GLOBAL_DMP300_V1_333
10-daily Gross Dry Matter Productivity 1KM	EO:CLMS:DAT:CGLS_GLOBAL_GDMP_V2_1KM
10-daily Gross Dry Matter Productivity 333M	EO:CLMS:DAT:CGLS_GLOBAL_GDMP300_V1_3
10-daily Water Bodies 1KM	EO:CLMS:DAT:CGLS_GLOBAL_WB_V2_1KM
10-daily Water Bodies 333M	EO:CLMS:DAT:CGLS_GLOBAL_WB300_V1_333M
10-daily Water Bodies over Africa 1km: Continents	EO:CLMS:DAT:CGLS_CONTINENTS_WB_V1_1KM
Agrometeorological indicators from 1979 to present derived from reanalysis	EO:ECMWF:DAT:SIS_AGROMETEOROLOGICAL_
Antarctic Ocean - Sea Ice Edge from SAR	EO:MO:DAT:SEAICE_ANT_SEAICE_L4_NRT_OBS
Arctic Ocean - High resolution Sea Ice Concentration and Sea Ice Type	EO:MO:DAT:SEAICE_ARC_PHY_AUTO_L4_NRT_(

Showing 1 to 10 of 1,148 entries Previous 1 2 3 4 5 ... 115 Next

It's now possible to obtain the `datasetId` directly from the dynamic table above.

✔ Now that you know how to look for your data of interest, you are ready to start accessing and downloading the products!

➤ 4. Data Access, Preview and Download [Back to top](#TOC_TOP)

As an initial example, you can use the dataset `ECMWF ERA5 Reanalysis Single Levels` which provides hourly estimates for many atmospheric, ocean and land variables.

The main goal is to understand how to retrieve and visualise a dataset of interest within the Notebook using the **HDA API**.

The first step is to store the `datasetId` of the selected dataset into a string variable, as follows:

In [8]:
```
dataset_id = 'EO:ECMWF:DAT:REANALYSIS_ERA5_SINGLE_LEVELS'
```

Data and metadata preview

To start exploring a dataset, it is always suggested to access all the descriptive information (metadata) and generate a graphical preview of it.

By re-using the previous strategy (i.e. using a Pandas DataFrame with itables), it is possible to select only the dataset you are interested in - in this case, the `ERA5 Reanalysis Single Levels` - by filtering the table on the `datasetId`.

In [ ]:
```
size = 2000
dataset = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/databr
data = json.loads(dataset.text)
data_df = pd.json_normalize(data['content'])
data_df = data_df[data_df['datasetId'] == dataset_id]
show(data_df, scrollX=True, columnDefs=[{"className": "dt-left", "targets": "_all"
```

You have now the full information on the dataset in different columns of the Pandas DataFrame:

- `abstract` : a description of the dataset
- `datasetId` : the id of the dataset, previously described
- `previewImage` : a link to a preview image of the dataset
- `title` : the title of the dataset

You can use the dataset `title`, `preview image` and `abstract` to create a dynamic widget using ipywidgets library, thus enhancing the exploration of the dataset.

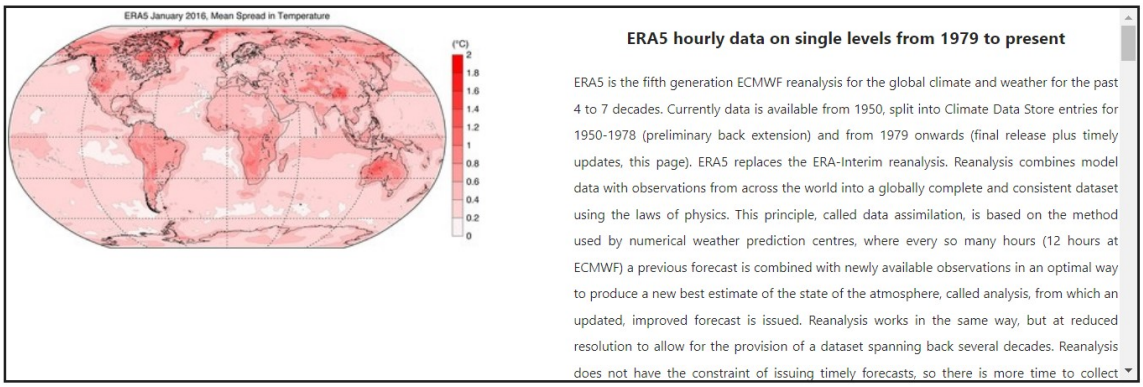
To do so, you need to create the variables to feed the widget as shown below.

In [10]:
```
# Get the dataset title from data_df
title = data_df.title.values[0]

# Get the dataset abstract from data_df
description = list(data_df["abstract"])[0]

# Get the dataset preview mage preview and create a display image using IPython
img_url = list(data_df["previewImage"])[0]
image = IPython.display.Image(img_url, width = 500)
image = widgets.Image(value = image.data,format="jpg", width=500,height=600)
```

Now, let's use the ipywidgets `HTML` and `Box` controls to create the graphical interface for the visualization. You can customize controls' styles and spacing as you wish.

In [ ]:
```
# Create the boxes
title_box = widgets.HTML('<h2 style="text-align:center;font-size:18px;">' + title
descr_box = widgets.HTML('<p style="text-align:justify;font-size:14px;">' + descri
image_box = widgets.VBox([image]) #image box
descr_box = widgets.VBox([title_box, descr_box]) #description box
```

```
# Create the Layout for the dataset preview
ui = widgets.AppLayout(right_sidebar = descr_box, left_sidebar = image_box, layout

# Create and display the container
container = widgets.Box([ui], layout = Layout(height = '400px', overflow_y = 'auto
display(container)
```



Set up an interactive dashboard to automate data requests

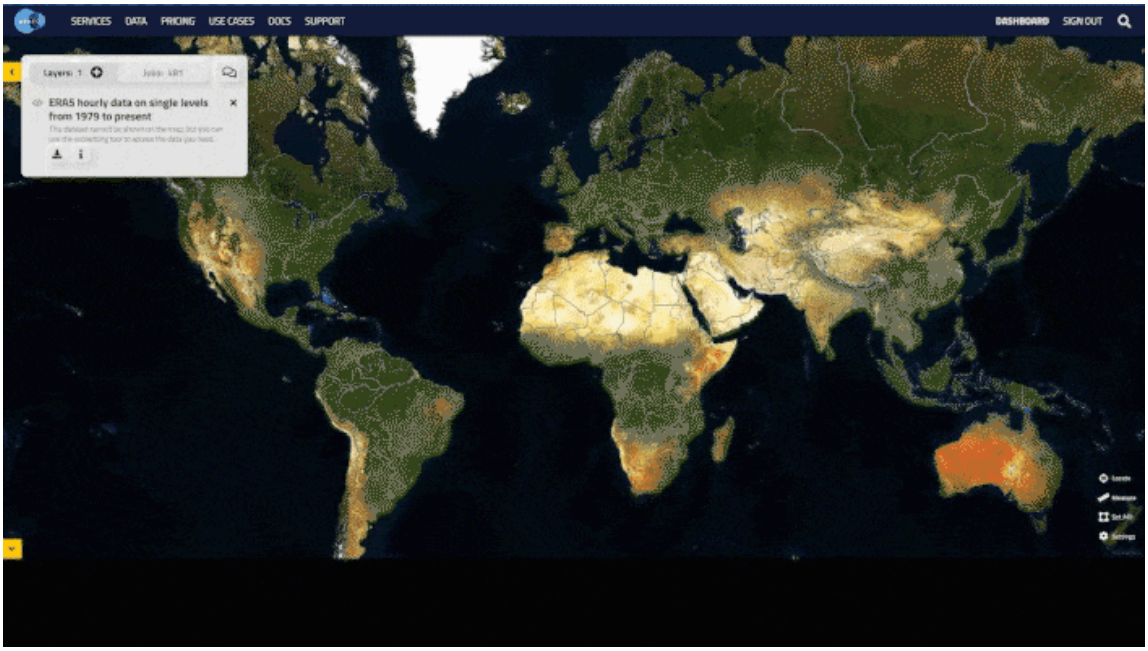
For `ERA5 Hourly Data On Single Levels` you have to provide the following information about the data you want to request:

- `Variable` : 'product variable (atmospheric/climate)'
- `Format type` : 'e.g. NetCDF'
- `Product type` : 'e.g. ensemble_mean'
- `Year`
- `Month`
- `Day`
- `Time` : 'UTC hours'

All the parameters must be provided as *strings* (or a *list of strings*).

Let's try now to create an interactive dashboard to automate data requests.

On the WEkEO Data Discovery Platform, it is possible to check the parameters needed to perform the API request for accessing any dataset (and show each API request).



Accessing metadata

To build the dashboard, you need to access the dataset's `metadata`. The metadata can be requested in JSON format and they contain all the information necessary to create the widgets, and finally the dashboard.

The metadata can be obtained using the following `GET` request:

In [12]:
```
headers = {'authorization': token}
dataset = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/databro
metadata = json.loads(dataset.text)
```

We can interactively visualize the metadata using the IPython `display` function. This is very useful to understand how the `JSON` file containing the dataset metadata is structured and how it can be exploited to create the API requests. Try to open the metadata file and figure out how the dataset is structured:

In [ ]:
```
display(JSON(metadata))
```

Create the widgets

In order to create dynamic widgets for accessing the data, you have first to extract values from the `metadata` file, convert them into *lists of strings* and store them into Python variables.

For `ERA5 Hourly Data On Single Levels` dataset, these operations can be performed as follows:

In [14]:
```
category = metadata['parameters']['multiStringSelects'][0]['details']['groupedValu
```

```
category_list = []
params_list = []
for item in category:
    category_list.append(item['valuesLabels'])

for item in category_list:
    key_list = list(item.keys())
    params_list.append(key_list)

variables_list = [item for sublist in params_list for item in sublist]
# variables_list   # Uncomment to show the variables list
```

⚠️ The previous step may not be necessary for other types of datasets. For the ERA5 Hourly Data On Single Levels, this is required since the atmospheric variables are grouped under multiple labels.

The other necessary parameters can be directly extracted as follows:

In [15]:
```
format_type_list = list(['netcdf'])
product_type_list = list(metadata['parameters']['multiStringSelects'][1]['details']
year_list = list(metadata['parameters']['multiStringSelects'][2]['details']['groupe
month_list = list(metadata['parameters']['multiStringSelects'][3]['details']['grou
day_list = list(metadata['parameters']['multiStringSelects'][4]['details']['groupe
time_list = list(metadata['parameters']['multiStringSelects'][5]['details']['groupe
```

Now that you have all the parameters stored in Python variables, you can create a simple interactive dashboard to explore and access the data products.

In this example, a multiple selection widget (`SelectMultiple`) is used for all parameters, becouse you may need to select multiple variables or time periods for accessing a specific resource within the dataset. Check the IPyWidgets documentation to get more information on all the available widgets.

In [16]:
```
style = {'description_width': '200px'}
layout = {'width': '800px'}
params_sel = widgets.SelectMultiple(options = variables_list, description = 'Variab
product_type_sel = widgets.SelectMultiple(options = product_type_list, description
year_sel = widgets.SelectMultiple(options = year_list, description = "Year: ", disa
month_sel = widgets.SelectMultiple(options = month_list, description="Month: ", dis
day_sel = widgets.SelectMultiple(options = day_list, description = "Day: ", disable
time_sel = widgets.SelectMultiple(options = time_list, description = "Time: ", disa
format_type_sel = "netcdf"  #For simplicity only NetCDF files are considered (also
```

A widget for each variable is finally created, providing the corresponding list to the `options` parameter and assigning a custom styling or layout.

Create the dashboard and request the data

You can now assemble the widgets in a single dashboard. The widgets are grouped using again the IPyWidgets `VBox` container.

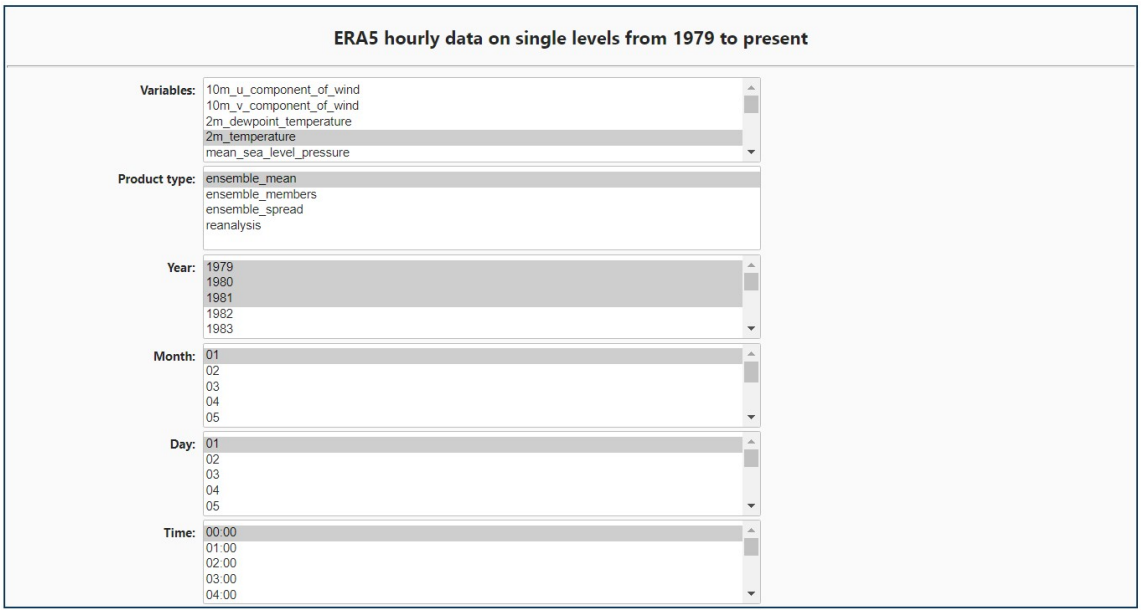
Some simple `HTML` / `CSS` code is also applied to improve the appearance of the dashboard. The following cell can be used to define CSS attributes and customize the styling. You can add additional `HTML` / `CSS` code to adapt the visualization to your needs.

In [ ]:
```
%%html
<style>
.box {
    border: 2px solid #0b385f;
    font-weight: bold;
    background-color: hsl(0, 0%, 98%);
    color: #333;
}
```

Let's test the dashboard by selecting `2m_temperature` as `Variable` and `ensemble_mean` as `Product type`. You can select the other parameters as you like. You can select multiple values by holding down `Ctrl` while selecting.

> ⚠️ Selecting long time periods or many variables may result in long-running API requests. For testing purposes, just select only a few data.

In [ ]:
```
# Create boxes for the dashboard
title_box= widgets.HTML(value='<h2 style="text-align:center;">'+title+'</h2><hr>')
variables_box = widgets.VBox([title_box, params_sel, product_type_sel, year_sel, m

# Create the dashboard
ui = widgets.AppLayout(
          layout=widgets.Layout(),
          grid_gap="300px")

# Display the variable box
display(variables_box.add_class("box"))
```



Changing the parameter selection in the dashboard above will automatically change the requested data in future API requests. The API `POST` request forward the following JSON-like variables to the server for retrieving data.

The dashboard is used to interactively modify the API requests which are actually run by the two next code blocks.

After the request is completed it returns a `Job ID` that will allow you to download the data through a subsequent `GET` request.

Notice also that the `headers` variable need to be modified.

In [19]:
```
query = {
  "datasetId": dataset_id,
  "multiStringSelectValues": [
    {
      "name": "variable",
      "value": list(params_sel.value)
    },
    {
      "name": "year",
      "value": list(year_sel.value)
    },
    {
      "name": "month",
      "value": list(month_sel.value)
    },
    {
      "name": "day",
      "value": list(day_sel.value)
    },
    {
      "name": "time",
      "value":list(time_sel.value)
    },
    {
      "name": "product_type",
      "value": list(product_type_sel.value)
    }
  ],
  "stringChoiceValues": [
    {
      "name": "format",
      "value": format_type_sel
    }
  ]
}

headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
    'authorization': 'Basic ' + str(token)
}

data = json.dumps(query)
dataset_post = requests.post("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/
job_id = json.loads(dataset_post.text)
jobId = job_id['jobId']
print("The job ID is: " + jobId)
```

```
The job ID is: KfqQhR2ursL8d--gjccPJYsz5mo
```

Once the `Job ID` is displayed from the previous code block, you can check if the request is `Completed`.

In [20]:
```
get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu/dat
get_url = json.loads(get_url_request.text)
if get_url['status'] == 'completed':
    print('Status: Completed', end = '\r')  #if status is completed

while get_url['status'] != 'completed':
    get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu
    get_url = json.loads(get_url_request.text)
```

```
    if get_url['status'] == 'running':   #if status is running
        print('Status: Running', end = '\r')
    elif get_url['status'] == 'failed': #if status is failed
        print('Status: Failed. Check data selected.')
        break
    elif get_url['status'] == 'completed': #if status is completed
        print('Status: Completed')
```

```
Status: Completed
```

⚠️ You must wait until the request is **Completed**. If an error occurs, check if the parameters are selected correctly inside the dashboard, or if the token is still valid.

If no errors appear, you are ready to perform a `GET` request to download the selected data. The URL for the download will be generated and displayed.

In [21]:
```
headers = {'authorization': 'Basic ' + str(token)}
get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu/dat
get_url = json.loads(get_url_request.text)
url = get_url['content'][0]['url']
print('The URL for download is: ' + get_url['content'][0]['url'])
```

```
The URL for download is: https://download-0018.copernicus-climate.eu/cache-compute
-0018/cache/data6/adaptor.mars.internal-1658756138.970197-29498-9-af948918-5837-4f
64-bf4d-2c9549fceb65.nc
```

✔️ You have obtained the URL needed to download the data resource from the HDA API!

The code can be adapted to all WeKEO datasets with a few modifications (see sections 5 and 6)

Download or read the data

In principle, you can download the data on your disk and then import it into the Notebook to continue with your analysis. To avoid moving data in and out of the Notebook, an option is open the data resource provided by the HDA API directly in the computer memory.

The `ERA5 Hourly Data On Single Levels` data format is the **NetCDF**. The same format is used for many of the Copernicus data distributed by WEkEO.

The following `RadioButtons` widget is used to specify the downloading option.

- **Download** the NetCDF file in the working directory and then read it
- **Read** the NetCDF in-memory without downloading it

In [ ]:
```
download_list = [ "Read NetCDF in memory", "Download NetCDF"] #Choice - Download d
download_sel = widgets.RadioButtons(options = download_list, description = "Data d
download_sel
```

The `download_type` function is defined inside the wekeo2pydash_methods.py file, and it uses the xarray library to handle NetCDF.

Depending on the option selected in the previous widget, it is possible to download the dataset (*the NetCDF file will be placed in the same folder of the Notebook*) or directly read the NetCDF in memory without any local file downloaded.

In [23]:
```
ds = m.download_type(download_sel, download_list, get_url) #use the "download_type
ds #display the raw NetCDF
```

Out[23]: xarray.Dataset

► Dimensions: (**longitude**: 720, **latitude**: 361, **time**: 3)

▼ Coordinates:

longitude	(longitude)	float32	0.0 0.5 1.0 ... 35...
latitude	(latitude)	float32	90.0 89.5 89.0 .....
time	(time)	datetime64[ns]	1979-01-01 19...

▼ Data variables:

t2m	(time, latitude, longitude)	float32	...

▼ Attributes:

Conventions :	CF-1.6
history :	2022-07-25 13:35:39 GMT by grib_to_netcdf-2.25.1: /opt/ecmwf/mars-client/bin/grib_to_netcdf.bin -S param -o /cache/data6/adaptor.mars.internal-1658756138.970197-29498-9-af948918-5837-4f64-bf4d-2c9549fceb65.nc /cache/tmp/af948918-5837-4f64-bf4d-2c9549fceb65-adaptor.mars.internal-1658756138.0958364-29498-12-tmp.grib

Interactive visualization

To complete your data exploration, you can create an interactive plot with the NetCDF file previously downloaded. For example, by using a `Dropdown` widget and passing as `option` the time coordinate of your data, you will be able to display interactively your variables in different periods.

⚠ To exploit the time coordinates in the plot, you have to download a dataset covering two or more years, months and/or hours.

In [24]:
```
# Select the NetCDF variables
variables = list(ds.keys())
var_drop = widgets.Dropdown(options = variables, description = "Variable: ", style

# Select the NetCDF times
timings = list(ds.time.data)
time_drop = widgets.Dropdown(options = timings, description = "Select date: ", dis
```

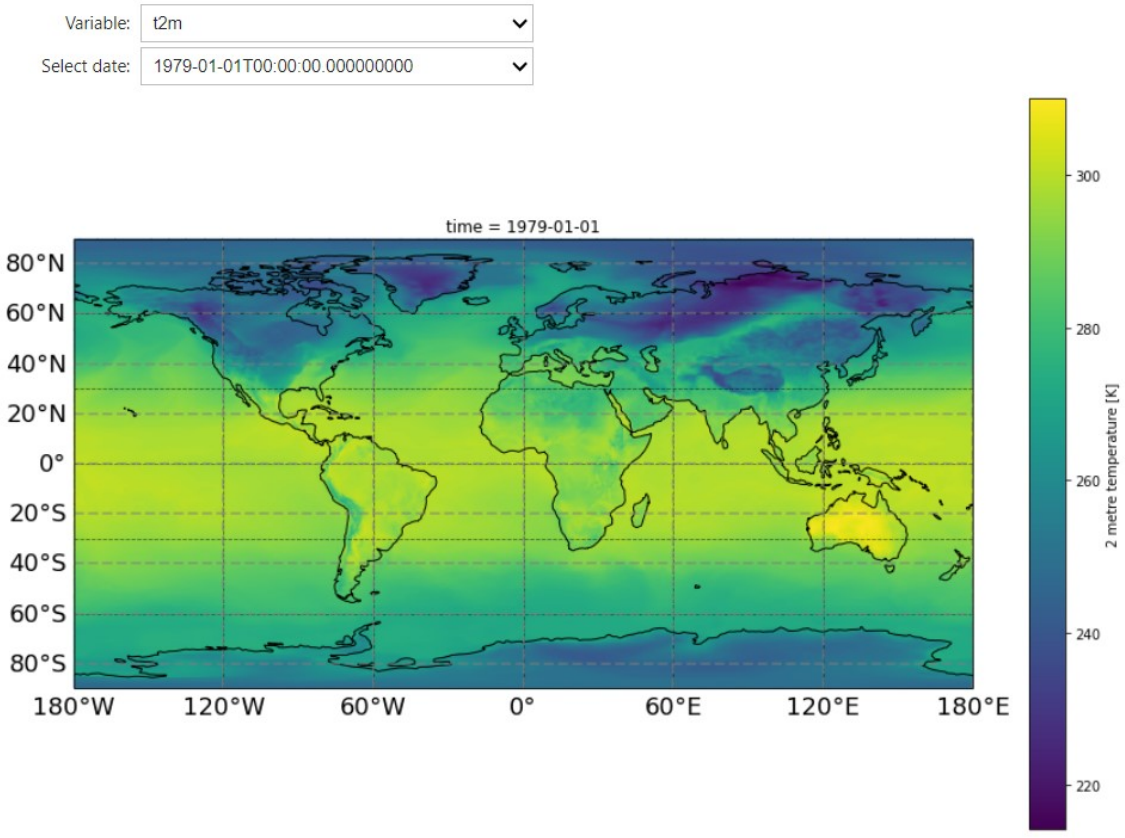
The interactive plot function exploits the `interact` widget's functionality to update the

visualization when a parameter is changed from the dropdown menu.

> ⚠ The generation of the plot at each widget interaction may take some second.

In [25]:
```
def plot_era5(variable, time):
    f = plt.figure(figsize = (15,10))
    p = ds[var_drop.value].sel(time = time_drop.value).plot.pcolormesh(  #change t
        subplot_kws = dict(projection = ccrs.PlateCarree(), facecolor="gray"),
        transform = ccrs.PlateCarree())
    p.set_clim(ds[var_drop.value].min(), ds[var_drop.value].max())
    p.axes.set_global()  #global
    p.axes.coastlines()  #coastlines
    p.axes.gridlines(color = 'black', alpha = 0.5, linestyle = '--')
    p.axes.set_extent([-180, 180, -90, 90], ccrs.PlateCarree()) #extent window

    # draw gridlines
    gl = p.axes.gridlines(crs = ccrs.PlateCarree(), draw_labels = True,
                          linewidth = 2, color = 'gray', alpha = 0.5, linestyle =

    # adjust labels
    gl.xlabels_top = False
    gl.ylabels_right = False
    gl.ylocator = mticker.AutoLocator()
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    gl.xlabel_style = {'size': 18, 'color': 'black'}
    gl.ylabel_style = {'size': 18, 'color': 'black'}
```

In [ ]:
```
widgets.interact(plot_era5, variable = var_drop, time = time_drop)
```



> ✔ You have successfully run the whole workflow presented in this Notebook for Copernicus data access, browse, display and download using the WeKEO HDA API!

⚠ The workflow is repeated and adapted to other datasets in the following sections. Instructions and comments will be provided only for those steps which require substantial modification than the one used for the ERA5 Hourly Data On Single Levels dataset.

➤ 5. Example A - CAMS - Europe Air Quality Forecasts [Back to top](#TOC_TOP)

In this section, the Notebook is adapted to the `European Air Quality Forecast` dataset provided by the Copernicus Atmosphere Monitoring Service (CAMS).

Differently from ERA5 Hourly Data On Single Levels dataset, the WeKEO HDA API allow filtering of the `European Air Quality Forecast` dataset also based on a Region of Interest (`Bounding Box`).

In running this section, you will go over many of the steps done in the previous chapter, and it will be shown how code blocks can be adapted to request the `European Air Quality Forecast` data.

⚠ This example relies on the Library imported in Section 1 and the Login procedure explained in 2. If you have not done that already, you have to run these two sections before proceeding with this example.

In this case, the WEkEO `datasetId` will be the following.

In [27]:
```
dataset_id = 'EO:ECMWF:DAT:CAMS_EUROPE_AIR_QUALITY_FORECASTS'
```

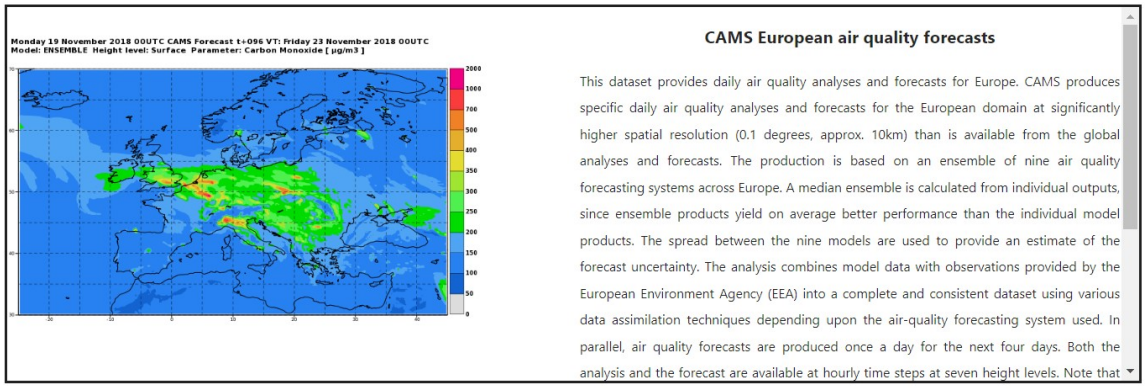
Data and metadata preview

In [ ]:
```
size = 2000
dataset = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/databro
data = json.loads(dataset.text)
data_df = pd.json_normalize(data['content'])
data_df = data_df[data_df['datasetId'] == dataset_id]
show(data_df, scrollX=True, columnDefs = [{"className": "dt-left", "targets": "_all
```

In [29]:
```
# Get the dataset title from data_df
title = data_df.title.values[0]

# Get the description from data_df
description = list(data_df["abstract"])[0].replace("", "\u03BC")

# Get the dataset image preview and create a display
img_url = list(data_df["previewImage"])[0]
```

```
image = IPython.display.Image(img_url, width = 500)
image = widgets.Image(value = image.data,format = "jpg", width=500,height=600)
```

In [ ]:
```
# Create the boxes
title_box = widgets.HTML('<h2 style="text-align:center;font-size:18px;">' + title
descr_box = widgets.HTML('<p style="text-align:justify;font-size:14px;">' + descri
image_box = widgets.VBox([image])
descr_box = widgets.VBox([title_box, descr_box])

# Create the Layout for the dataset preview
ui = widgets.AppLayout(right_sidebar = descr_box, left_sidebar = image_box,grid_ga
container = widgets.Box([ui], layout = Layout(height = '400px', overflow_y = 'auto
display(container)
```



Set up an interactive dashboard to automate CAMS European Air Quality Forecast data requests

For `CAMS European Air Quality Forecast` you have to provide the following information:

- `Bounding Box` : bounding box latitude and longitude (west, east, north, south)
- `Date Range` : start and end date
- `Variable` : product variable (pollutants)
- `Model` : air quality models (e.g. ensemble)
- `Level` : meters above surface
- `Type` : analysis or forecast
- `Time` : model base time
- `Lead time Hour` : forecast lead time in hours
- `Format` : e.g. NetCDF

Accessing metadata

In [ ]:
```
dataset = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/databro
metadata = json.loads(dataset.text)
display(JSON(metadata))
```

Create the widgets

In [32]:
```
format_type_list = list(['netcdf'])
params_list = list(metadata['parameters']['multiStringSelects'][0]['details']['grou
product_type_list = list(metadata['parameters']['multiStringSelects'][1]['details'
level_list = list(metadata['parameters']['multiStringSelects'][2]['details']['grou
type_list = list(metadata['parameters']['multiStringSelects'][3]['details']['groupe
hour_list = list(metadata['parameters']['multiStringSelects'][4]['details']['groupe
leadtime_list = list(metadata['parameters']['multiStringSelects'][5]['details']['gr
```

Create the widgets for the interactive dashboard. We use a `SelectMultiple` widget for all the variables except for the date, for which we use a `DatePicker` widget.

Check the IPyWidgets documentation to get more information on all the available widgets.

In [33]:
```
style = {'description_width': '200px'}
layout = {'width': '800px'}

params_sel = widgets.SelectMultiple(options = params_list, description = 'Variables
product_type_sel = widgets.SelectMultiple(options = product_type_list, description
level_sel = widgets.SelectMultiple(options = level_list, description = 'Level: ', 
type_sel = widgets.SelectMultiple(options = type_list, description = 'Type: ', disa
hour_sel = widgets.SelectMultiple(options = hour_list, description = 'Hour: ', disa
leadtime_sel = widgets.SelectMultiple(options = leadtime_list, description = 'Leadt
format_type_sel = "netcdf"
start_date_sel = widgets.DatePicker(description='Select start date: ', disabled = F
end_date_sel = widgets.DatePicker(description='Select end date: ', disabled = Falsе
```

Bounding Box interactive selection

As mentioned at the beginning of this section, for the `CAMS European Air Quality Forecast` dataset you can limit the data request to a Region of Interest by specifying a `Bounding Box` within the HDA API call.

The ipyleaflet library provides a solution to define the `Bounding Box` interactively through a simple web map interface.

The function `draw_map` is available in the wekeo2pydash_methods.py file. The following code block will generate a map and by using the `Draw a rectangle` tool (*on the left side of the map panel*) the `Bounding Box` area can be defined.

> ⚠️ As for the other widgets, also for the map you just need to run the code once and draw the Bounding Box without running the code block again, just jump to the next.

In [ ]:
```
cams_map, dc = m.draw_map(50, 10, 2)  # center lat=45, lon=10, zoom=2
area_extent = Rectangle(bounds=((30, -25), (72, 45)), color='red', fill_color='red
cams_map.add_layer(area_extent)
cams_map
```

⚠️ The red canvas in the map identifies the maximum spatial coverage of the CAMS European Air Quality Forecast dataset

The `Bounding Box` coordinates are obtained automatically from the rectangle you drew.

You are allowed to manually define `W`, `E`, `N`, `S` variables by passing numeric (Float) *Latitude* and *Longitude* values of the Bounding Box corners to select a specific Region of Interest (instead of drawing it).

In [35]:
```
coords = dc.last_draw['geometry']['coordinates'][0]
W = coords[1][0]
E = coords[3][0]
N = coords[1][1]
S = coords[3][1]
```

It's also suggested to check the time availability of the data before running the download request.

In [36]:
```
start_date = metadata['parameters']['dateRangeSelects'][0]['details']['start']
print("The start date for this dataset is: " + start_date + ". You can select data
```

```
The start date for this dataset is: 2019-03-26. You can select data after this dat
e.
```

Create the dashboard and request the data

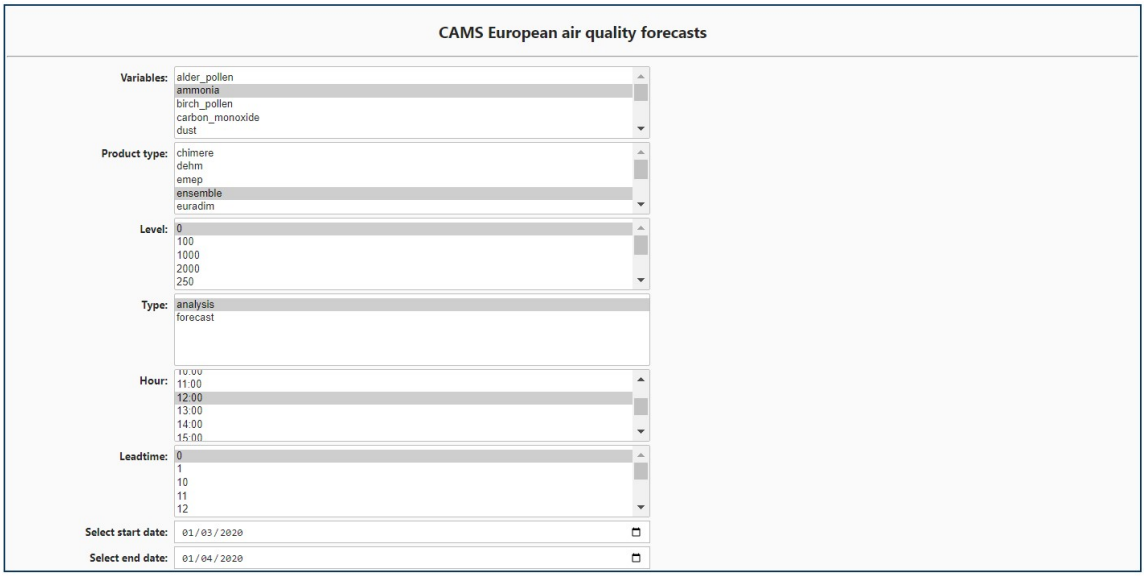
You can now create a dashboard (as you did in the previous section). This time you will use the widgets specifically created for the `CAMS European Air Quality Forecast` dataset. Try to select the following parameters to run this example:

- `variable` : ammonia
- `product type` : ensemble
- `level` : 0
- `type` : analysis
- `leadtime hour` : 0
- `hour` : 12:00
- `start-end date` : from 01/03/2020 to 01/04/2020

In [ ]:
```
%%html
<style>
.box {
```

```
    border: 2px solid #0b385f;
    font-weight: bold;
    background-color: hsl(0, 0%, 98%);
    color: #333;
}
```

In [ ]:
```
# Create boxes for the dashboard
title_box= widgets.HTML(value='<h2 style="text-align:center;">' + title + '</h2><h
variables_box = widgets.VBox([title_box, params_sel, product_type_sel, level_sel, 

# Create the dashboard
ui = widgets.AppLayout(layout=widgets.Layout(), grid_gap="300px")

display(variables_box.add_class("box"))
```



After the selection of the parameter values on the dashboard, you can run the HDA API request.

In [39]:
```
query = {
      "datasetId": dataset_id,
      "boundingBoxValues": [
        {
          "name": "area",
          "bbox": [
            W,
            N,
            E,
            S
          ]
        }
      ],
      "dateRangeSelectValues": [
        {
          "name": "date",
          "start": start_date_sel.value.strftime("%Y-%m-%dT%H:%M:%S.000Z"),
          "end": end_date_sel.value.strftime("%Y-%m-%dT%H:%M:%S.000Z")
        }
      ],
      "multiStringSelectValues": [
        {
          "name": "variable",
          "value": list(params_sel.value)
        },
```

```
        {
          "name": "model",
          "value": list(product_type_sel.value)
        },
        {
          "name": "level",
          "value": list(level_sel.value)
        },
        {
          "name": "type",
          "value": list(type_sel.value)
        },
        {
          "name": "time",
          "value": list(hour_sel.value)
        },
        {
          "name": "leadtime_hour",
          "value": list(leadtime_sel.value)
        }
      ],
      "stringChoiceValues": [
        {
          "name": "format",
          "value": format_type_sel
        }
      ]
    }

headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
    'authorization': 'Basic ' + str(token)}

data = json.dumps(query)
dataset_post = requests.post("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/
job_id = json.loads(dataset_post.text)
jobId = job_id['jobId']
print("The job ID is: " + jobId)
```

```
The job ID is: -PE9GEDzhPQvlzQJMp3SnvPRE9Y
```

Once the `Job ID` is displayed from the previous code block, you can check if the request is `Completed`.

In [40]:

```
get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu/dat
get_url = json.loads(get_url_request.text)
if get_url['status']=='completed':
    print('Status: Completed', end='\r')

while get_url['status']!='completed':
    get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu
    get_url = json.loads(get_url_request.text)
    if get_url['status']=='running':
        print('Status: Running', end='\r')
    elif get_url['status']=='failed':
        print('Status: Failed. Check data selected.')
        break
    elif get_url['status']=='completed':
        print('Status: Completed')
```

```
Status: Completed
```

⚠ You must wait until the request is **Completed**. If an error occurs, check if the parameters are selected correctly inside the dashboard, or if the token is still valid.

If no errors appear, you are ready to perform a `GET` request to download the selected data. The URL for the download will be generated and displayed.

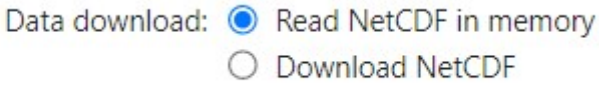
In [41]:
```
headers = {'authorization': 'Basic ' + str(token)}
get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu/dat
get_url = json.loads(get_url_request.text)
# print(get_url)
url = get_url['content'][0]['url']
print('The URL for download is: ' + get_url['content'][0]['url'])
```

```
The URL for download is: https://download-0002-ads-clone.copernicus-climate.eu/cac
he-compute-0002/cache/data3/adaptor.cams_regional_fc.retrieve-1658756218.1413226-5
978-11-668f8960-8ff7-4cbd-b115-c9022fd7733f.nc
```

Download or read the data

You can now decide whether you want to download the `NetCDF` file with its original name or read it directly in memory.

In [ ]:
```
download_list = ["Read NetCDF in memory", "Download NetCDF",] #Choice - Download d
download_sel = widgets.RadioButtons(options = download_list, description = "Data d
download_sel
```



In [43]:
```
ds = m.download_type(download_sel, download_list, get_url)
ds
```

Out[43]: xarray.Dataset

► Dimensions: (**longitude**: 84, **latitude**: 39, **level**: 1, **time**: 32)

▼ Coordinates:

longitude	(longitude)	float32	7.05 7.15...
latitude	(latitude)	float32	47.45 47....
level	(level)	float32	0.0
time	(time)	timedelta64[ns]	0 days 1...

▼ Data variables:

nh3_conc	(time, level, latitude, longitude)	float32	...

▼ Attributes:

title :	NH3 Air Pollutant ANALYSIS at the Surface
institution :	Data produced by Meteo France
source :	Data from ENSEMBLE model
history :	Model ENSEMBLE ANALYSIS
ANALYSIS :	Europe, 20200301-20200401+[12H_12H]
summary :	ENSEMBLE model hourly ANALYSIS of NH3 concentration at the Surface from 20200301-20200401+[12H_12H] on Europe
project :	MACC-RAQ (http://macc-raq.gmes-atmosphere.eu)

NetCDF time format conversion

Different from the ERA5 Hourly Data On Single Levels dataset, the `time` variable for the `CAMS European Air Quality Forecast` is encoded as a timedelta (i.e. the difference between two times). To that end, **you need to change it from `timedelta` to `datetime` to obtain the actual time** to which the data refer to.

In [44]:
```
#Define times
timestamp = ds.time.long_name[19:27]
time_start= int(hour_sel.value[0][0:2])  #if using data starting at different hour
timestamp_init=datetime.datetime.strptime(timestamp,'%Y%m%d') + datetime.timedelta
time_coords = pd.date_range(timestamp_init, periods = len(ds.time), freq = '1d').st

# Assign the datetimes instead of timedeltas
ds_assign = ds.assign_coords(time = time_coords)
ds_assign = ds_assign.assign_coords(longitude = (((ds_assign.longitude + 180) % 360
ds_assign
```

Out[44]: xarray.Dataset

► Dimensions: (**longitude**: 84, **latitude**: 39, **level**: 1, **time**: 32)

▼ Coordinates:

longitude	(longitude)	float32	7.05 7.15 ...
latitude	(latitude)	float32	47.45 47....
level	(level)	float32	0.0
time	(time)	datetime64[ns]	2020-03-...

▼ Data variables:

nh3_conc	(time, level, latitude, longitude)	float32	...

▼ Attributes:

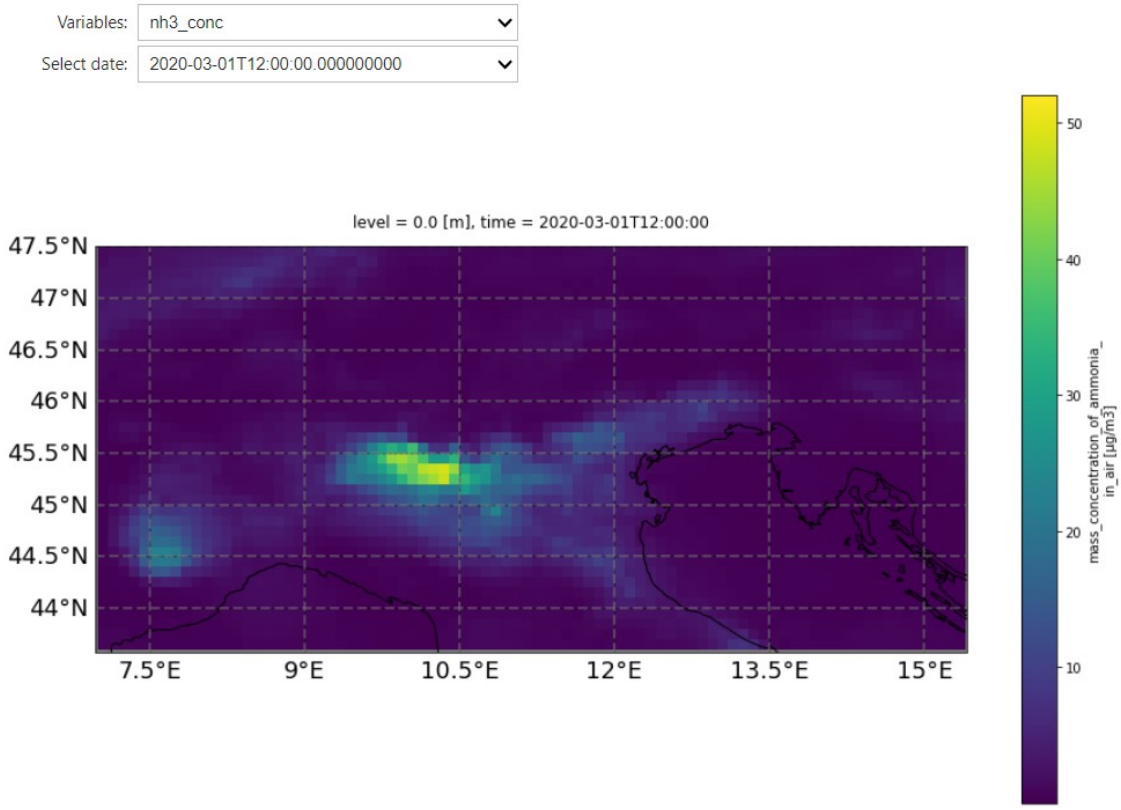
title :	NH3 Air Pollutant ANALYSIS at the Surface
institution :	Data produced by Meteo France
source :	Data from ENSEMBLE model
history :	Model ENSEMBLE ANALYSIS
ANALYSIS :	Europe, 20200301-20200401+[12H_12H]
summary :	ENSEMBLE model hourly ANALYSIS of NH3 concentration at the Surface from 20200301-20200401+[12H_12H] on Europe
project :	MACC-RAQ (http://macc-raq.gmes-atmosphere.eu)

Interactive visualization

In [45]:
```
# Select the netCDF variables
variables = list(ds.keys())
var_drop = widgets.Dropdown(options = variables, description = 'Variables: ', disal
# Select the netCDF times
timings = list(ds_assign.time.data)
time_drop = widgets.Dropdown(options = timings, description = "Select date: ", dis
```

In [46]:
```
def cams_plot(variable, time):
    f = plt.figure(figsize = (15,10))
    p = ds_assign[var_drop.value].sel(time = time_drop.value).plot(  #change time
    subplot_kws = dict(projection = ccrs.PlateCarree(), facecolor="gray"),
    transform = ccrs.PlateCarree())
    p.set_clim(ds_assign[var_drop.value].min(),ds_assign[var_drop.value].max())
    p.axes.set_global()
    p.axes.coastlines()
    p.axes.gridlines(color = 'black', alpha = 0.5, linestyle = '--')
    p.axes.set_extent([W, E, S, N], ccrs.PlateCarree())

    gl = p.axes.gridlines(crs = ccrs.PlateCarree(), draw_labels = True,
                          linewidth = 2, color = 'gray', alpha = 0.5, linestyle =

    gl.xlabels_top = False
    gl.ylabels_right = False
    gl.ylocator = mticker.AutoLocator()
    gl.xformatter = LONGITUDE_FORMATTER
    gl.yformatter = LATITUDE_FORMATTER
    gl.xlabel_style = {'size': 18, 'color': 'black'}
    gl.ylabel_style = {'size': 18, 'color': 'black'}
```

In [ ]:
```
widgets.interact(cams_plot, variable = var_drop, time = time_drop)
```

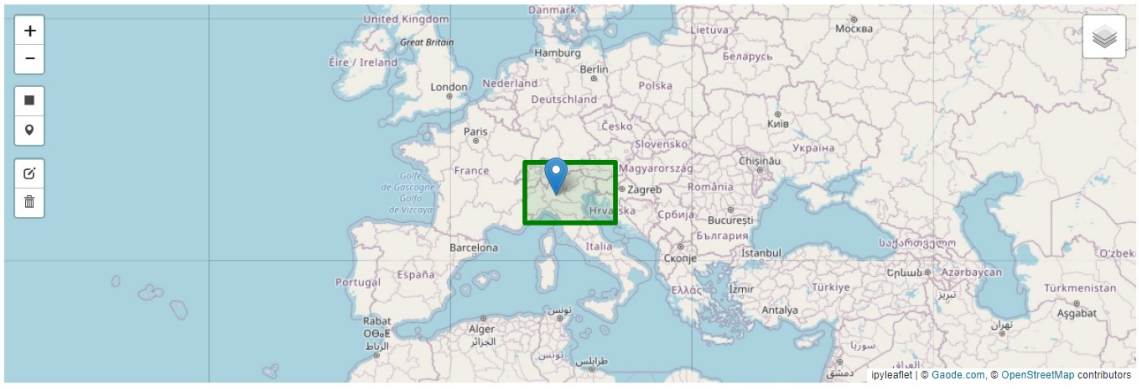


Interactive time-series plotting in a specific location

The ipyleaflet functionalities can be also used to select a specific location to sample the corresponding time series from the `CAMS European Air Quality Forecast` variable observations.

You can plot the map again and, by using this time the `Draw a marker` tool (*on the left side of the map*), you can select **a location within the `Bounding Box` that you have previously drawn**.

In [ ]:
```
cams_map, dc = m.draw_map((N + S) / 2,(E + W) / 2, 4)
area_extent = Rectangle(bounds=((S, W), (N, E)), color='green', fill_color='green'
cams_map.add_layer(area_extent)
cams_map
```



⚠️ You must select a location within the data Bounding Box because the map function exploits the same NetCDF which you download in the previous steps. The green canvas

on the map identifies this area.

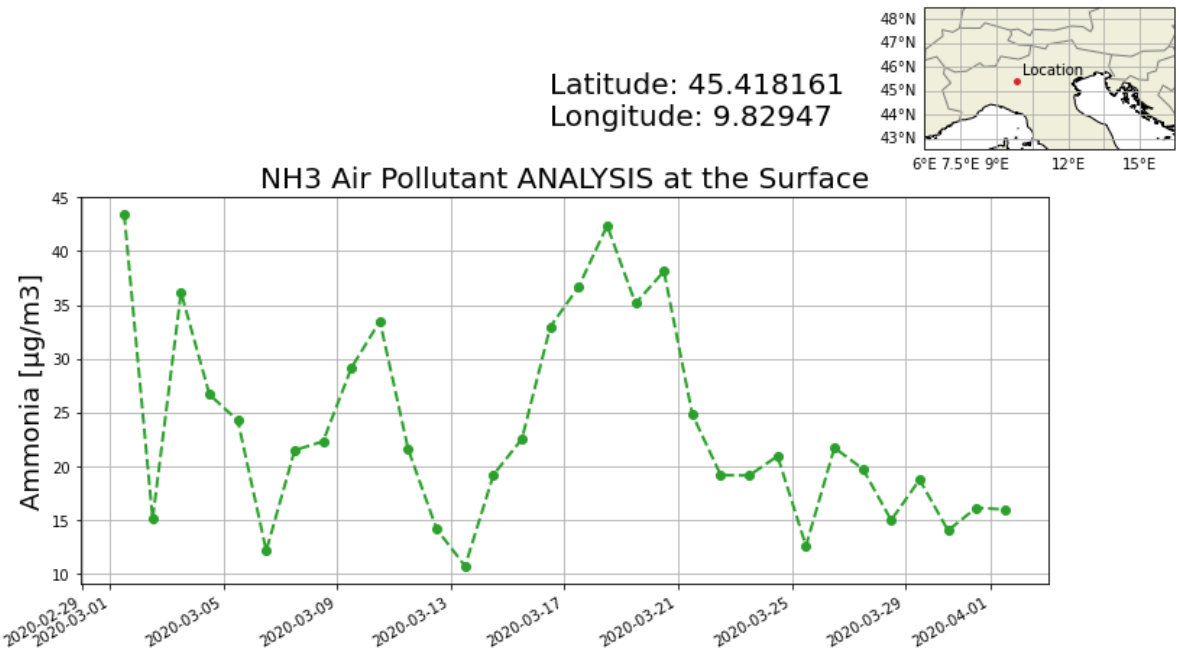
The `longitude` and the `latitude` of the location are obtained automatically from the *marker* you drew.

You can now plot the time serie for the selected variable and location:

In [49]:
```
coords = dc.last_draw['geometry']['coordinates']
lati=coords[1]
loni=coords[0]

var = ds_assign.sel(longitude = loni, latitude = lati, method = 'nearest')[var_drop
```

In [50]:
```
f = plt.figure(figsize = (12,12))
ax=f.add_subplot(211)

ax.set_title(ds_assign.title , fontsize = 20)
ax.grid()
ax.set_ylabel(ds_assign[var_drop.value].species + ' ' + '['+ds_assign[var_drop.valu
ax.plot(ds_assign.time, var,c = 'tab:green', linewidth = 2, marker = 'o', linestyle
f.suptitle('Latitude: ' + str(lati) + ' \nLongitude: ' + str(loni),ha = 'left', fo
f.autofmt_xdate()

states_provinces = cfeature.NaturalEarthFeature(
        category='cultural',
        name='admin_0_boundary_lines_land',
        scale='110m',
        facecolor='none')
ax_mini_map = f.add_axes([0.8, 0.90, 0.2, 0.15], projection = ccrs.PlateCarree())
gl = ax_mini_map.axes.gridlines(draw_labels = True)
ax_mini_map.add_feature(states_provinces, edgecolor = 'gray')
gl.xlabels_top = False
gl.ylabels_right = False
ax_mini_map.add_feature(cfeature.LAND, zorder = 0, edgecolor='k')
ax_mini_map.set_extent([W-1, E+1, S-1, N+1])
ax_mini_map.scatter(loni, lati, 15, 'tab:red', transform = ccrs.PlateCarree())
ax_mini_map.annotate('Location', (loni, lati), xytext = (4, 4), textcoords ='offse
```

Out[50]:
```
Text(4, 4, 'Location')
```



✔ You have successfully run the Notebook workflow for Copernicus data access, browse, display and download using the WeKEO HDA API, adapted and extended to the CAMS European Air Quality Forecast dataset!

➤ 6. Example B - Sentinel-5P - Air quality [Back to top](#TOC_TOP)

In this section, the Notebook is adapted to the `Sentinel-5P` dataset provided by the Sentinel-5P Copernicus mission, dedicated to atmospheric constituents monitoring at a global scale.

In running this section, you will go over many of the steps done in the previous chapter, and it will be shown how code blocks can be adapted to request the `Sentinel-5P` data.

⚠ This example relies on the Library imported in Section 1 and the Login procedure explained in 2. If you have not done that already, you have to run these two sections before proceeding with this example.

In this case will, the WEkEO `datasetId` be the following.

In [51]:
```
dataset_id = 'EO:ESA:DAT:SENTINEL-5P:TROPOMI'
```

Data and metadata preview

In [ ]:
```
size = 2000
dataset = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/databr
data = json.loads(dataset.text)
data_df = pd.json_normalize(data['content'])
data_df = data_df[data_df['datasetId'] == dataset_id]
show(data_df, scrollX=True, columnDefs=[{"classname": "dt-left", "targets": "_all"
```

Search:

	abstract
1012	The Sentinel-5 Precursor (Sentinel-5P) polar orbiting satellite is an active European Space Agency Earth observation platform that is funded under the European Commission's Copernicus Earth Observation Programr

Showing 1 to 1 of 1 entries

For the `Sentinel-5P` dataset, the HDA API does not provide a preview image, and you can only read the dataset description.

In [53]:
```
# Get the dataset title from data_df
title = data_df.title.values[0]
# No image for S5P
# Get the description from data_df
description = list(data_df["abstract"])[0]
```

In [ ]:
```
# Create the boxes
title_box = widgets.HTML('<h2 style="text-align:center;font-size:18px;">'+title+'</
descr_box = widgets.HTML('<p style="text-align:justify;font-size:14px;">'+descript

descr_box = widgets.VBox([title_box, descr_box])

# Create the Layout for the dataset preview
```

```
ui = widgets.AppLayout(right_sidebar=descr_box, layout=widgets.Layout(border='soli
container = widgets.Box([ui], layout=Layout(height='300px', overflow_y='auto'))
display(container)
```

SENTINEL-5 TROPOMI

The Sentinel-5 Precursor (Sentinel-5P) polar orbiting satellite is an active European Space Agency Earth observation platform that is funded under the European Commission's Copernicus Earth Observation Programme. It orbits the Earth 14 times a day. Sentinel-5P carries the TROPOspheric Monitoring Instrument (TROPOMI) spectrometer. This instrument senses ultraviolet (UV), visible (VIS), near (NIR) and short-wave infrared (SWIR). Output from these sensors are used to generate products to monitor ozone, methane, formaldehyde, aerosol, carbon monoxide, nitrogen dioxide and sulphur dioxide in the atmosphere. These products are known as level 2 products. TROPOMI takes measurements covering an area of 2600km by 7km. The satellite was launched in October 2017 and entered into routine operational phase in March 2019. Data is available from July 2018 onwards. The Sentinel-5P programme is an active and developing mission, from time to time ESA reprocesses data in order to take into account new scientific insights and operational developments.

Set up an interactive dashboard to automate Sentinel-5P data requests

For `Sentinel-5P` you have to provide the following information:

- `Bounding Box` : Region of Interest
- `Start-end date`
- `Processing level` : Level 1B/Level 2
- `Product type` : the type of product, such as NO2 or other pollutants
- `Timeliness` : Near Real Time, Offline, Reprocessing

Accessing metadata

In [ ]:
```
dataset = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/databro
metadata = json.loads(dataset.text)
display(JSON(metadata))
```

Create the widgets

In [56]:
```
# Create the list of variables
processing_level_list = list(metadata['parameters']['stringChoices'][0]['details']
product_type_list = list(metadata['parameters']['stringChoices'][1]['details']['va
timeliness_list = list(metadata['parameters']['stringChoices'][2]['details']['valu
```

Create the widgets for the interactive dashboard. You can use the `SelectMultiple` widget for all the parameters except for the date, for which you can use the `DatePicker` widget.

Check the IPyWidgets documentation to get more information on all the available widgets.

In [57]:
```
style = {'description_width': '200px'}
layout = {'width': '800px'}
processing_level_sel = widgets.SelectMultiple(options = processing_level_list, des
product_type_sel = widgets.SelectMultiple(options = product_type_list, description
timeliness_sel = widgets.SelectMultiple(options = timeliness_list, description = "
start_date_sel = widgets.DatePicker(description = "Select start date: ", disabled
end_date_sel = widgets.DatePicker(description = "Select end date: ", disabled = Fa
```

Bounding Box interactive selection

The `Sentinel-5P` dataset data request you can be searched in Region of Interest by specifying a `Bounding Box` within the HDA API call.

The ipyleaflet library provides a solution to define the `Bounding Box` interactively through a simple web map interface.

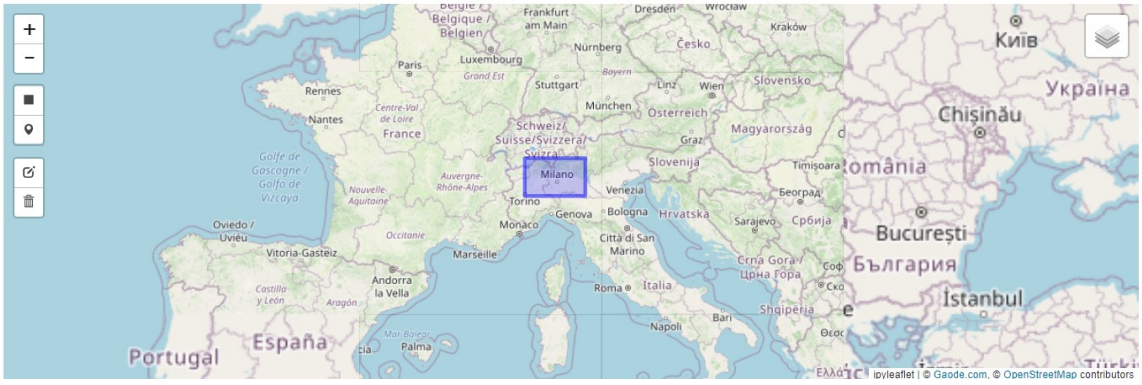
The function `draw_map` is available in the wekeo2pydash_methods.py file. The following code block will generate a map and by using the `Draw a rectangle` tool (*on the left side of the map panel*) the `Bounding Box` area can be defined.

Select the Region of Interest in the following map, always using the `Draw a rectangle`

⚠ The *Bounding Box* for the Sentinel-5P is not used to clip the requested data (as in Section 5) but it is used to retrieve all the satellite tiles touching the *Region of Interest*.

⚠ As for the other widgets, also for the map you just need to run the code once and draw the Bounding Box without running the code block again, just jump to the next.

In [ ]:
```
s5p_map, dc = m.draw_map(45, 10, 1)

s5p_map
```



In [60]:
```
coords = dc.last_draw['geometry']['coordinates'][0]
W = coords[1][0]
E = coords[3][0]
N = coords[1][1]
S = coords[3][1]
```

It's also suggested to check the time availability of the data before running the download request.

In [61]:
```
start_date = metadata['parameters']['dateRangeSelects'][0]['details']['start']
print("The start date for this dataset is: "+start_date+". You can select data afte
```

```
The start date for this dataset is: 2018-04-30T00:41:24Z. You can select data afte
r this date.
```

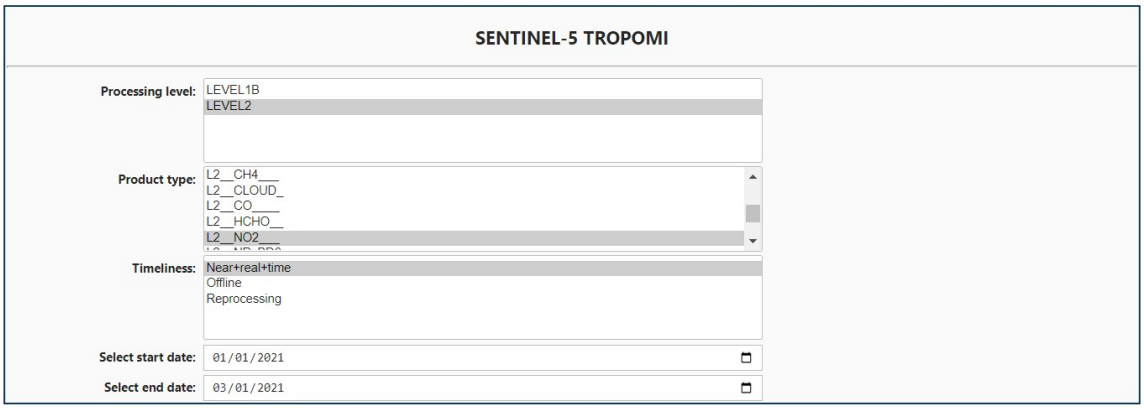
Create the dashboard and request the data

You can now create a dashboard (as you did in the previous sections). This time you will use the widgets created ad-hoc for the `Sentinel-5P` dataset. Try to select the following parameters to run this example.

- `Processing level` : Level2
- `Product type` : L2*NO2*
- `Timeliness` : Near+real+time
- `Start/end dates` : 01/01/2021 - 03/01/2021

In [ ]:
```
%%html
<style>
.box {
    border: 2px solid #0b385f;
    font-weight: bold;
    background-color: hsl(0, 0%, 98%);
    color: #333;
}
```

In [ ]:
```
# Create boxes for the dashboard
title_box= widgets.HTML(value='<h2 style="text-align:center;">' + title + '</h2><h
variables_box = widgets.VBox([title_box, processing_level_sel, product_type_sel, t

# Create the dashboard
ui = widgets.AppLayout(
          layout = widgets.Layout(),
          grid_gap = "300px")

display(variables_box.add_class("box"))
```



In [64]:
```
query = {
  "datasetId": dataset_id,
  "boundingBoxValues": [
    {
      "name": "bbox",
      "bbox": [W, S, E, N]
    }
  ],
  "dateRangeSelectValues": [
    {
      "name": "position",
      "start":  start_date_sel.value.strftime("%Y-%m-%dT%H:%M:%S.000Z"),
      "end": end_date_sel.value.strftime("%Y-%m-%dT%H:%M:%S.000Z")
    }
  ],
  "stringChoiceValues": [
    {
```

```
      "name": "processingLevel",
      "value": list(processing_level_sel.value)[0]
    },
    {
      "name": "productType",
      "value": list(product_type_sel.value)[0]
    },
    {
      "name": "timeliness",
      "value": list(timeliness_sel.value)[0]
    }
  ]
}

headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
    'authorization': 'Basic ' + str(token)}

data = json.dumps(query)
dataset_post = requests.post("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/
job_id = json.loads(dataset_post.text)
jobId = job_id['jobId']
print("The job ID is: " + jobId)
```

```
The job ID is: WfYEytPhQQxw80zlXjXkC9fOD5s
```

Once the `Job ID` is displayed from the previous code block, you can check if the request is `Completed`.

In [65]:
```
get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu/dat
get_url = json.loads(get_url_request.text)
if get_url['status'] == 'completed':
    print('Status: Completed', end = '\r')

while get_url['status'] != 'completed':
    get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu
    get_url = json.loads(get_url_request.text)
    if get_url['status'] == 'running':
        print('Status: Running', end = '\r')
    elif get_url['status'] == 'failed':
        print('Status: Failed. Check data selected.')
        break
    elif get_url['status'] == 'completed':
        print('Status: Completed')
```

```
Status: Completed
```

⚠️ You must wait until the request is **Completed**. If an error occurs, check if the parameters are selected correctly inside the dashboard, or if the token is still valid.

Read metadata and prepare Sentinel-5P data order

Differently from the `ERA5 Hourly Data On Single Levels` and the `CAMS European Air Quality Forecast` dataset, the HDA API requires to first place an order for the `Sentinel-5P` data products. This means that you have to specify the list of layers you want to access within the data product and then download them.

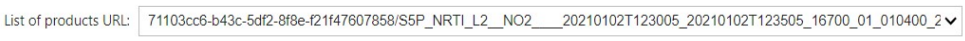
The HDA API provides a `JSON` file containing all the metadata associated with each product obtained from the request you run before through the dashboard.

In [66]:
```
headers = {'authorization': 'Basic ' + str(token)}
get_url_request = requests.get('https://wekeo-broker.apps.mercator.dpi.wekeo.eu/dat
get_url = json.loads(get_url_request.text)
```

In [ ]:
```
display(JSON(get_url['content']))
```

You can create a list of layers from the metadata `JSON` file and use a `Dropdown` widget to select one of them.

In [ ]:
```
s5p_list = []
for x in range(0, len(get_url['content'])):
    s5p_list.append(get_url['content'][x]['url'])

#Create widget for Sentinel-5P data selection
s5p_drop =widgets.Dropdown(options= s5p_list, description = "List of products URL:
s5p_drop
```



Order the data

You are now ready to order the specific layer you selected.

In [69]:
```
url = s5p_drop.value

headers = {
    'Content-Type': 'application/json',
    'Accept': 'application/json',
    'authorization': 'Basic ' + str(token)}

query = {
    "jobId":str(jobId),
    "uri":str(url)
    }

data = json.dumps(query)
dataset_post_order = requests.post("https://wekeo-broker-k8s.apps.mercator.dpi.weke
order_id = json.loads(dataset_post_order.text)['orderId']
print('The order ID is: ' + order_id)
```

```
The order ID is: nIGqzGTdNZWOl5plyHI0Sbqfo_8
```

The HDA API provides you with an `order ID` needed for the download. The `order ID` is displayed by the previous code block and stored automatically in a Python variable.

⚠ Wait until the **order ID** is displayed.

Download the data

The `order ID` variable is used to download the data through the following API `GET` request:

In [70]:
```
headers = {'Accept': 'application/zip'}

response_order = requests.get("https://wekeo-broker-k8s.apps.mercator.dpi.wekeo.eu/
```

The requested data in NetCDF format is obtained and it can be saved in the working directory (the same one containing the Notebook) as a compressed folder (*zip* format) using its original name.

In [71]:
```
zip_filename = url[url.find('/') + 1:]

with open(zip_filename, 'wb') as zfile:
    zfile.write(response_order.content)
zip_filename
```

Out[71]:
```
'S5P_NRTI_L2__NO2____20210102T123005_20210102T123505_16700_01_010400_20210102T132359.zip'
```

It's then possible to directly **uncompress the downloaded `Sentinel-5P` data folder** in the working directory.

In [72]:
```
with zipfile.ZipFile(zip_filename, 'r') as zip_ref:
    os.mkdir(zip_filename[0:-4])
    zip_ref.extractall()
```

And **select** the NetCDF file (*.nc*) contained in the uncompressed folder.

In [ ]:
```
path = "./" + zip_filename[0:-4]
nc_files = [f for f in os.listdir(path) if f.endswith('.nc')]
nc_drop = widgets.Dropdown(options=nc_files, description = "netCDF filename: ", dis
nc_drop
```

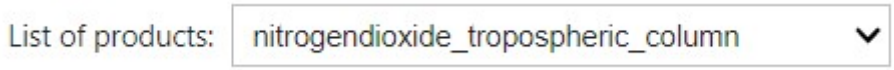
Read the NetCDF file using the netCDF4-Python library.

In [74]:
```
nc_file = path + "/" + nc_drop.value
fh = Dataset(nc_file, mode = 'r') #read NetCDF file
```

Data visualization

First, let's create the list of layers contained in the `Sentinel-5P` NetCDF file and select one of your interest. If you followed the instruction of the dashboard subsection, select the `nitrogendioxide_tropospheric_column` layer.

In [ ]:
```
product_list = list(fh.groups['PRODUCT'].variables.keys())
products_drop = widgets.Dropdown(options = product_list, description = "List of pr
products_drop
```



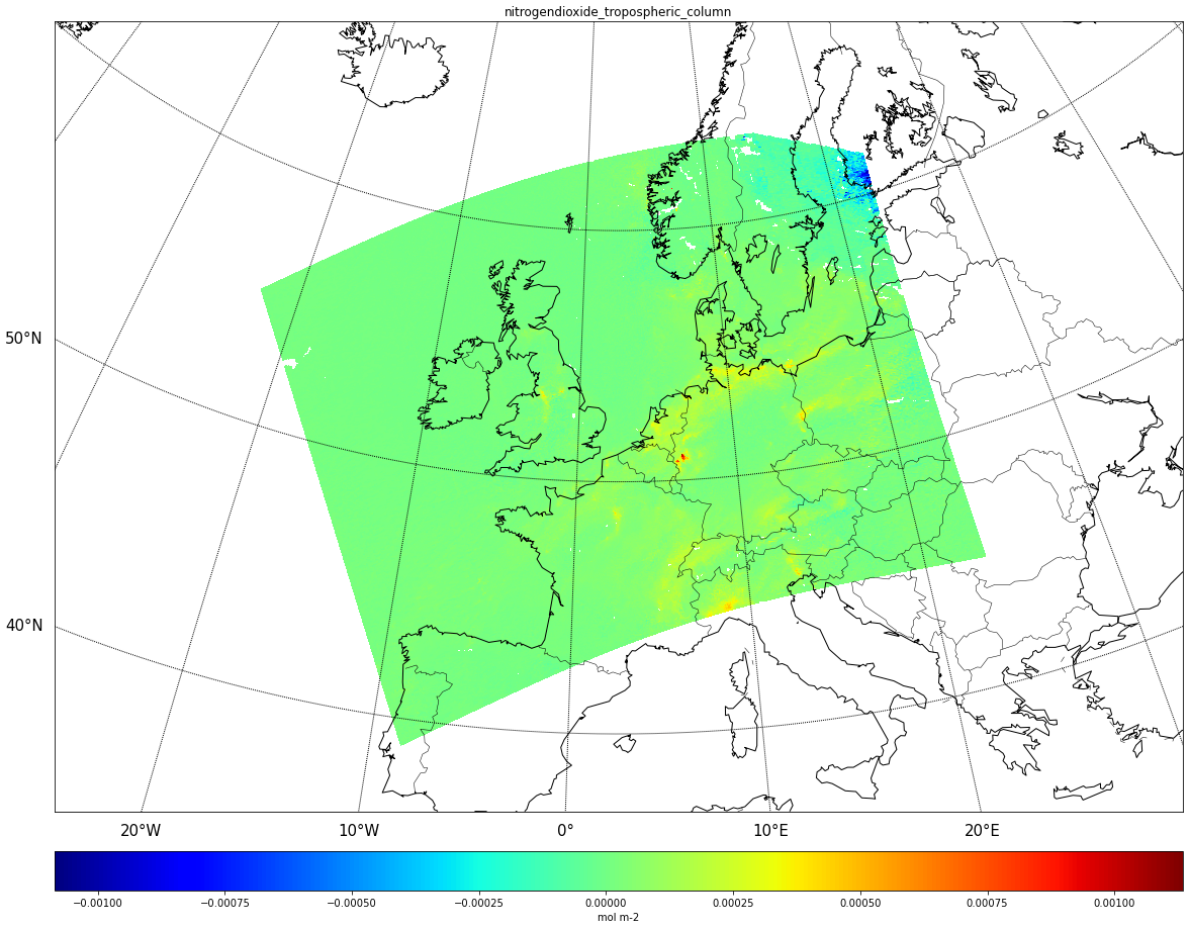
To dispaly the data on a map, you need to store the *latitude*, *longitude* and the grid values from the NetCDF file into the following Python variables.

In [76]:
```
lons = fh.groups['PRODUCT'].variables['longitude'][:][0,:,:]
lats = fh.groups['PRODUCT'].variables['latitude'][:][0,:,:]
prod = fh.groups['PRODUCT'].variables[products_drop.value][0,:,:]
print (lons.shape)  #check shape
print (lats.shape)
print (prod.shape)

units = fh.groups['PRODUCT'].variables[products_drop.value].units
```

```
(373, 450)
(373, 450)
(373, 450)
```

It's now possible to **plot** the data.

In [77]:
```
lon_0 = lons.mean()
lat_0 = lats.mean()
plt.figure(figsize=(20,20))
map1 = Basemap(width = 5000000, height = 3500000,
            resolution = 'l', projection = 'stere',\
            lat_ts = 40, lat_0 = lat_0, lon_0 = lon_0)

xi, yi = map1(lons, lats)

# Plot Data

cs = map1.pcolor(xi, yi, np.squeeze(prod), cmap='jet')

# Add Grid Lines
map1.drawparallels(np.arange(-80., 81., 10.), labels = [1,0,0,0], fontsize = 15)
map1.drawmeridians(np.arange(-180., 181., 10.), labels = [0,0,0,1], fontsize = 15)

# Add boundaries and coastlines
map1.drawcoastlines()
map1.drawstates()
map1.drawcountries()

# Cbar
cbar = map1.colorbar(cs, size = "5%", location = 'bottom', pad = "5%")
cbar.set_label(units)



# Title
plt.title(products_drop.value)
plt.show()
```



✔ You have successfully run the Notebook workflow for Copernicus data access, browse, display and download using the WEkEO HDA API, adapted and extended to the Sentinel-5P dataset!
