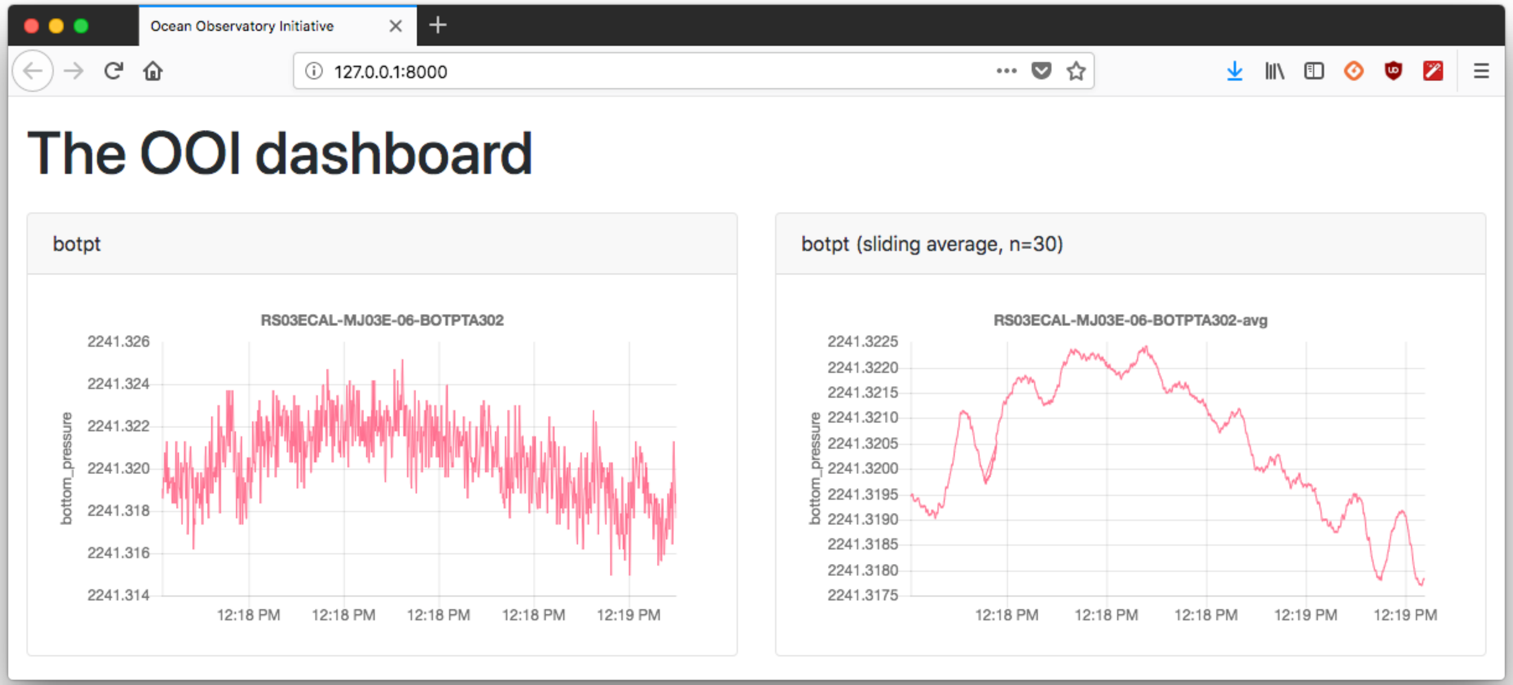
Diving into Big Data Workshop

Hands-on activity

During this activity, you will learn how to obtain live streaming data from the Ocean Observatory Initiative, how to plot it in a web dashboard and how to apply some transformations to the data as it is being continuously received.



Learn what are the available instruments with the interactive map:

https://goo.gl/JJyb7M

Download the starter code:

https://anthony-simonet.fr/ooi-starter-code.zip

Use users "workshop" with password "workshop2018".

Data format

Data in OOI is delivered in JSON, which means that each individual value is a JSON document similar to this:

```
{
      'time': 3727684681.9,
      'press_trans_temp': 3.21276087,
      ...
      'bottom_pressure': 2241.4755859375
}
```

`time` always represents the time when the measure was made, but all other keys depend on the particular instrument you are streaming from. On the interactive map, the list of available keys for each instrument is given under “variables”.

Overview of the code

Download and extract the starter code in `C:\Users\workshop\ooi-starter-code\` (right-click on the zip file then “Extract all”). To edit the code, you can use the **Rodeo IDE** (a shortcut is located on the Desktop).
The project includes a Django web server (for serving web pages) and a Kafka client (for consuming data streams from OOI). The code you received contains the following directories and files (only the most important ones are listed here, the ones you need to edit are **bold**):

```
ooi-starter-code/
      dashboard/
             config.py           // Server starter script
             settings.py         // Server setting
             urls.py             // URL mappings
             views.py            // Template rendering
      ooi/
             client.py
      static/                    // Static files for the client
             css/
             js/
      templates/
             index.html          // Client homepage
      manage.py
```

Instructions

Open a Windows command line

Your code will be executed indirectly by the Django web server in a command window. To get started:

- Open the Windows Start menu and run this command: `cmd`
- In the window that appears, change to the directory where the code is, e.g. `C:\Users\workshop\ooi-starter-code` by running:

```
cd ooi-starter-code
```

Start/stop the web server

To start the server, select the command window and execute:

```
python manage.py runserver
```

You can now view your dashboard by browsing http://localhost:8000

To stop the server, select the command window and press `Ctrl+Break`.

You will likely have to restart the server after making certain changes to your code.

Start consuming streaming data

Now that your server is running, you have to instantiate an OOIClient object. To do so, make the following changes to `dashboard/config.py`:

– Line 14, set the correct Kafka server address (`128.104.223.162:9092`);
– Line 16, set the topics you want to subscribe to as a `List` of `String` (see the map to learn about which topics are available: https://goo.gl/JJyb7M);
– After line 18, connect and start the server by calling the right methods on `self._client` (see `ooi/client.py` for available methods on `OOIClient`).

After restarting the web server, you should see the following messages on the console:

```
Worker thread started
```

Plot the stream on the client

You must now modify templates/index.html in order to add plots to http://localhost:8000. The page already includes JavaScript libraries that will automate the process of:

– Inserting a plot at a specified location in the page;
– Downloading the streaming data from the web server;
– Updating the plot with new data every few seconds.

Your first chart will be displayed in the `<canvas id="first-chart"></canvas>` element that is already present (and invisible for now) on the page. You can add several `<canvas>` elements to plot different topics.

The `id` attribute must be unique in the page!

To create the plot, look for the "`// Your code here`" comment and use the following JavaScript function (where `topic` and `variable` are taken from the map):

```
createGraph(element_id, topic, variable);
```

After saving the template file and reloading the page, you should see an empty plot for a couple seconds, then the actual plot with new data coming every 5 seconds.

Transform the data streams

For now, all your web server does is consume a stream from OOI and forward the raw data to clients. However, you can leverage all the power of Python to apply transformations to the streams.

To apply transformation to streams, you can provide as many handler functions as you wish to the `OOIClient`. The client will call these functions and pass them a `List` of data values every time new data is received from the OOI streams. The handler function must inspect the `List` of values and produce a new `List` object containing the transformed values.

The code in `dashboard/config.py` demonstrates a simple function that squares every value in a topic (see function `square`).

Using the following code, try and make a new handler function that computes the moving average of any topic and variable. Each point of a simple moving average is the average of the previous *n* points. This code computes a *centered moving average*, which is slightly more complex:

```
def slidingAverage(topic, startIndex):
    windowSize = 50
    half = int(windowSize / 2)

    data = OOIClient.data[topic]

    if not 'bottom_pressure' in data[0]:
        return []

    comp = OOIClient.data[topic + '-avg']
    res = []

    n = len(data)
    m = len(comp)

    if n < windowSize:
        return []

    start = max(m, half)
    end = n - half
    for i in range(start, end):
        winStart = i - half
        winEnd   = i + half

        sum = 0
        for i in range(winStart, winEnd):
            sum += data[i]['bottom_pressure']
        avg = sum / windowSize

        res.append({
            'time': data[i]['time'],
            'bottom_pressure': avg
        })

    return res
```

Don’t hesitate and try other transformations! Here are some more ideas:

- Convert temperatures from Celcius to Farenheit
- Remove outliers

APACHE kafka®
A distributed streaming platform

python

Rodeo

Chart.js

B
Bootstrap