# GrafanaTrackMap

July 10, 2021

## 1 Using TrackMap on Grafana



We expect that one has Grafana / IndluxDB / nginx enabled on a Docker environment as explained in the manual, chapter InfluxDB.

### 1.1 TrackMap installation

Plese refer first to plugin's pages.

The installations is done from command line with Grafana CLI in our standalone (local) system. On Windows, the easiest way to get it with with Docker Dashboard, navigate to the `dasht_grafana` container and click on the `CLI` button. Give command:

```
/usr/share/grafana $ grafana-cli plugins install pr0ps-trackmap-panel
installing pr0ps-trackmap-panel @ 2.1.2
from: https://grafana.com/api/plugins/pr0ps-trackmap-panel/versions/2.1.2/download
into: /var/lib/grafana/plugins

* Installed pr0ps-trackmap-panel successfully

Restart grafana after installing plugins . <service grafana-server restart>
```

As suggested, close the CLI windows, head back to the Docker Dashboard and restart the container `dasht_grafana`.

Using the Grafana administrator interface, you can now observe a new, signed plugin, `TrackMap`. When you create a panel, it shows up as an alternative `Visualization` method under the corresponding drop down menu. If it does not, you may need to restart the Grafana server, which can be used simply by first stopping and the starting the entire *DashT* Docker stack.

### 1.2 TrackMap and Flux

Apparently, TrackMap is a GIS application and only SQL is used. However, from the Grafana's point of view it is about the query and the fields it returns.

Somebody using TrackMap has had the issue already this year and come with a solution where the Flux itself has been used to map the fields the query produces. He came up with a solution like this:

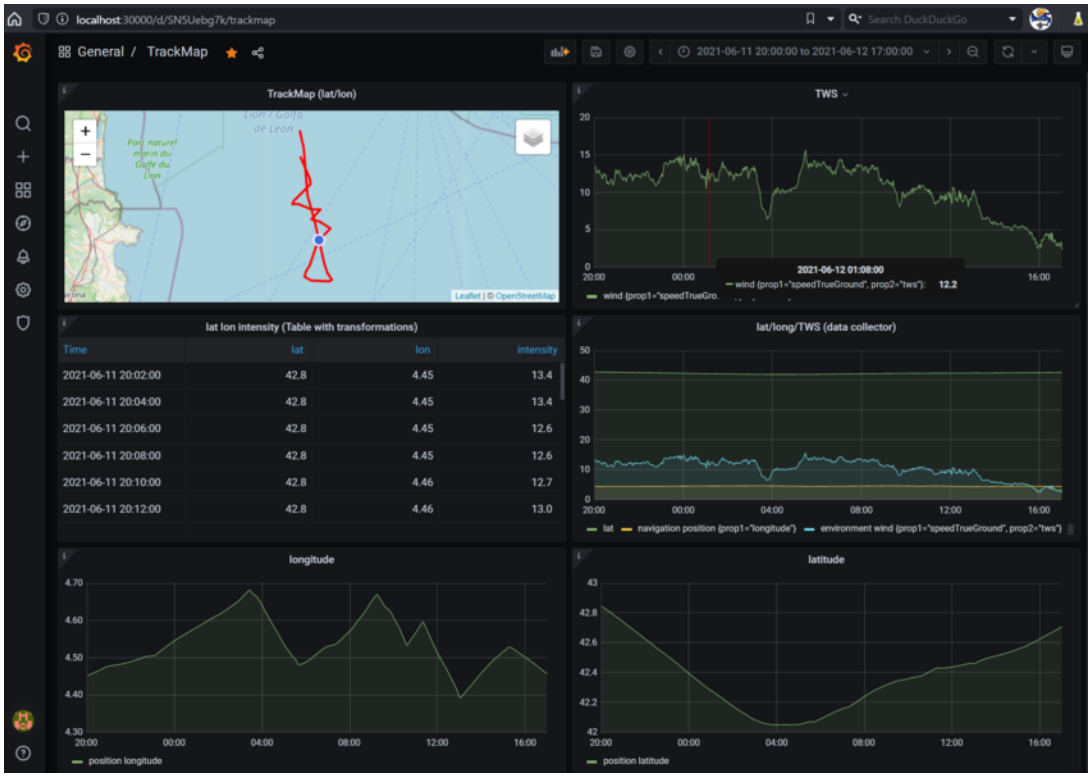
```
from(bucket: "home_assistant")
  |> range(start: v.timeRangeStart, stop: v.timeRangeStop)
```

```
  |> filter(fn: (r) => r["_measurement"] == "device_tracker.mate10")
  |> filter(fn: (r) => r["_field"] == "gps_accuracy" or r["_field"] == "latitude" or r["_fiel

|> pivot(columnKey: ["_field"], rowKey: ["_time"], valueColumn: "_value")

|> duplicate(column: "latitude", as: "lat")
|> duplicate(column: "longitude", as: "lon")
|> duplicate(column: "_time", as: "tooltip")
|> duplicate(column: "velocity", as: "popup")

|> map(fn: (r) => ({ r with popup: "Speed: " + string(v:r.popup) + " km/h" }))

|> keep(columns: ["_time", "tooltip", "lat", "lon", "popup"])
```

The above creates some errors in the InfluxDB query but provides `lat` and `lon` arrays alright in *DashT* use case with Grafana.

I rather make queries for all parameters in one single panel, where I can observe the field results, say in a Table visualization. Then, in the TrackMap panel I would read from this panel and use Flux Transformation functions I can select which queries are actually executed and give new names to the fields, etc.

Here is the Grafana 7.5.7 JSON-model to be imported as a Dashboard. This the same example dashboard used in making the TrackMap example in the User's Manual, it looked like this in its total size. Note that only one panel is making data inquiries (multiple) from the database, others are using and transforming that inquiry inside Grafana:



(zoom)

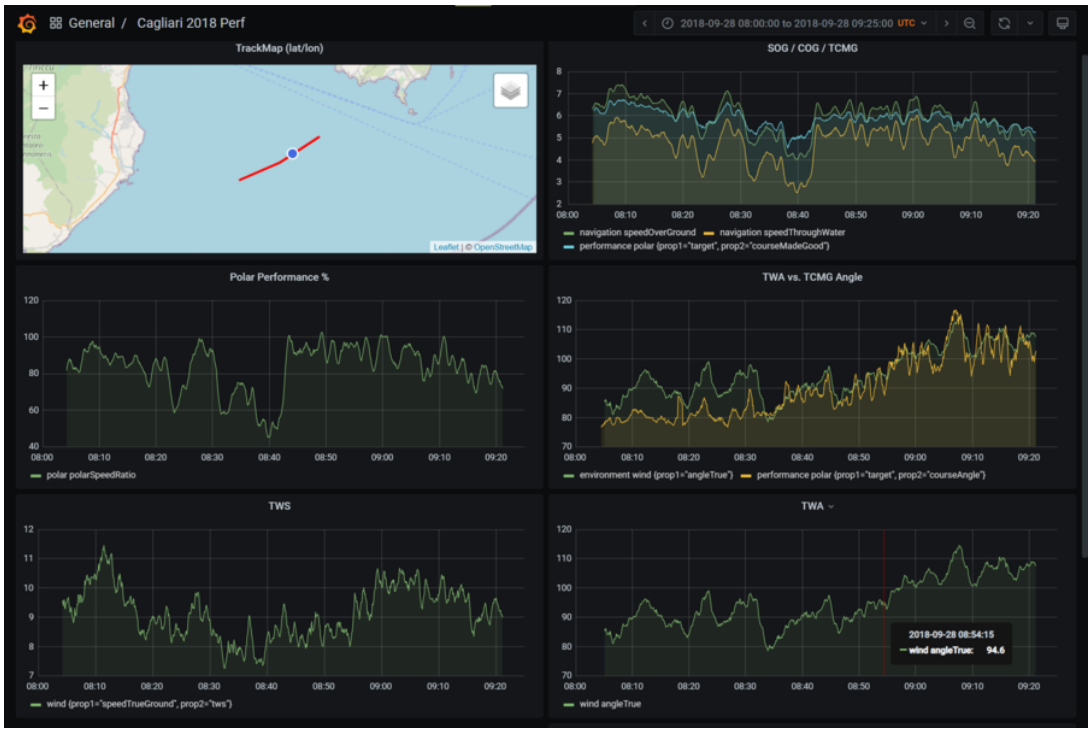
### 1.2.1 More complex example

The above dashboard with TrackMap and a centralized, single panel making all queries, one per data parameter, which can be distributed to other panels, selection taking place in the

Transformations post-processing.

The data is also historical, from 2018, and input from a VDR record file but with sufficient timestamps from GPS allowing the approximation of the timestamps in between the GPS ticks using the CPU clock. With Grafana level display, or even when zooming in with InfluxDB, there is is no discontinuity visible.

Here is the Grafana 7.x JSON-model of the Dashoard making the panels below. The data is from my personal historical roadbooks. Like in the previous example, there is a single panel hidden below which sole usage is to make all data inquiries from the database while the other, performance oriented databases are requesting the data internally, within Grafana.



(zoom)

### 1.3 Alexandra Institute's Track Map

Since we dropped on this track map as well, let's record here the experience: It looks promising with its heat map - one could show the intensity of the wind, or the polar performance!

There is a zip/tar.gz distribution of this more complete (and more complex) GIS tool.

Somebody using Alexandra Track Map has had the issue already this year and come with a solution where the Flux itself has been used to map the fields the query produces. He came up with a solution with pure Flux filtering (see the link).

I tried it with Alexandra Track Map v2.2.1, some adaptation as from the Data Explorer of InfluxDB copying the first part for the query, but in vain.

It is noteworthy that Alexandra Track Map does v2.2.1 does not deal with the Grafana transform functions but it must be the flat query which needs to provide all columns. This makes it quite complicated with Flux to get them joined: the resulting two tables are well understood by other modules, but not this one.

Others have had this issue as well and provided a pull request to overcome this by providing 'customs' field names for *lat* and *lon*, see `options.coordinates` in GrafanaAlexsandra-TrackMapCustom.json. I installed this version but it would work only with SQL, with Flux

there are two tables always underneath and since this plugin cannot deal with *join* transformation by time, it is quite hopeless since the above *pivot()* table solution does not work, either.