Kpler Challenge

Ship to Ship Cargo Transfer
Oil product storage pipeline Architecture

François Ferrari
07/June/2020

1 Ship To Ship Cargo Transfer

1.1 Introduction

The goal of this challenge is to identify the ships who are proceeding to ship to ship transfers, given a file containing the positions of ships for a period of time. Ship to Ship transfer, as described below, is the process of moving a product (oil, petrol ...) carried by a ship (A) to another ship (B) while still staying away from a port (harbour).

We will base our analysis on the pdf file provided by Kpler for this challenge (referenced as **KPDF**), please read this document to better understand the challenge.

1.1.1 Basic concepts

To better understand the basic concepts related to STS it is important to look at the videos whose links are provided in KPDF.

On top of this the following resources can be useful:

Concepts	**Resource**
Draught & Keel	https://www.youtube.com/watch?v=YpCyP1sM_cA
Course & Bearing	https://www.youtube.com/watch?v=ZVduCCEWiLc
Course & Bearing	https://en.wikipedia.org/wiki/Course_(navigation)#:~:text=In%20navigation%2C%20the%20course%20of,bow%20or%20nose%20is%20pointed.
Lightering	https://teekay.com/wp-content/uploads/2014/11/Lightering101.pdf

1.2 Analysis

The dataset we have to use to identify the STS cargo transfer is linked in KPDF, after downloading this file we can see that it is not too big, so we will be able to open this file with a simple editor, manipulate it with the standard linux tools, and to use Spark to provide some statistics about the different fields. The dataset is a typical csv file, the field separator is a comma.

1.2.1 Dataset content

Below is a description of the fields found in this dataset, and how useful they will be to solve the challenge or not.

1.2.1.1 Id

This is the Kpler's internal identifier for the **event** (or position).

The value is of type long.

We will keep track of this field in the data model so that we or other modules can eventually use it later.

1.2.1.2 Vessel id

This is the identifier of the vessel for the AIS event.

This is an essential value as it will allow us to produce an STS event for which two vessels are involved, and we want to know who are this vessels (we have a database of vessels, their ids are constant, we wan't to keep track of the habit of a vessel to categorize it - STBL, Daughter Vessel)

The value is of type long.

We will keep this field in the model.

1.2.1.3 Latitude & Longitude

This is the latitude and longitude of the event as computed by the AIS for this event.

The value is of type double.

We will keep this 2 values in the model as they are one of the components that will allow us to idenfity an STS event (distance between ships)

1.2.1.4 Received time

This is the timestamp of the emission of the event by the AIS.

The value is of type timestamp (ex: 2016-12-31 14:00:15) and it represents a UTC time.

We will keep this event in the model as it is one of the components that will allow us to identify an STS event (vessels being close to each other in a specific time window)

1.2.1.5 Speed, Course, Heading

This three values represent respectively the speed, course and heading of the vessel. Heading is not always valued, but we will use this value instead of course to understand if the vessels are going in the same direction.

The value for speed and course is of type double.

The value for the heading is of type integer.

We will keep this three values as they are one of the components that will allow us to idenfity an STS event (vessels should move at a very similar speed and heading during an STS event).

1.2.1.6 Draught

This is the distance from waterline to keel of a vessel and this is manually set by the crew.

The variation of the draught could be useful to understand if a vessel becomes lighter or heavier, thus validating our intuition about an STS event.

But looking at the dataset the draught value doesn't change during STS events.

We won't keep this value in the model as it doesn't look reliable.

1.2.1.7 Provider id

This is the AIS signal provider’s id in Kpler's database.

This may be used to understand how accurate an AIS event is.

We won't keep this value in the model as we can't really use it to identify an STS event.

1.2.1.8 AIS type

This allows to know if the AIS signal comes from a radio antenna or a satellite.

We won't keep this value in the model.

1.2.1.9 Added at, Added by, Updated At, Updated By

This fields are Kpler's internal data related to when and by which process this data/row was added or last updated into Kpler's database.

We won't keep this values as they are not useful to identify an STS event and they are still available in Kpler's position database based on the "id" field of the event.

1.2.1.10 Old last representation

This value is not described in KPDF, it looks to be a boolean in the dataset, and its value is always false.

We won't keep this field in the model as it doesn't seem useful to identify an STS event.

1.2.1.11 Point

This field is computed from the latitude and longitude to facilitate computations.

We won't keep this field in the model as we are not using it to compute the distance between the vessels (we use latitude and longitude instead), but if we were to use a specialized library that would use this type of value or if the consumers of our STS model need it, then we would bring it back in the model.

1.2.1.12 New navigation Status

This field contains a Navigational Status, this is manually set by the crew.

Some values could be interesting to identify STS events.

We could validate some of this values based for example on the vessel speed (under way sailing, moored).

We won't keep this field in the model.

1.3 Identifying STS events

The dataset being a time series dataset of vessel movements on the sea, it would be good to have a visual representation of this dataset so that we get a better intuition about the data.

We are going to use QGIS to project the events on a map and visualize the vessel tracks.

But we encounter two problems:

1. the dataset contains records for all the vessels, but we would like to be able to differentiate the vessels tracks in QGIS, hiding/showing them, this is why we will split the dataset in as many files as there are vessel ids in the dataset (there are 43 different vessels and it is manageable in GQIS).
2. the dataset records are not sorted by receive time, so we will need to sort the dataset using this field.

1.3.1 Sorting the dataset

We will sort the data using the received_time_utc field and we will use this sorted dataset from now on. The following command can be used to sort the dataset:

```
mv Galveston_2017_11_25-26.csv Galveston_2017_11_25-26.csv_orig
head -1 Galveston_2017_11_25-26.csv_orig > Galveston_2017_11_25-26.csv
tail +2 Galveston_2017_11_25-26.csv_orig | sort -t, -k5 >> Galveston_2017_11_25-26.csv
rm Galveston_2017_11_25-26.csv_orig
```

1.3.2 Splitting the data into smaller files per vessel_id

The command below allows to split the original dataset in as many files as there are vessel ids in the dataset, the files will be named "*vesselid*.csv".

```
dataset=Galveston_2017_11_25-26.csv
header=`head -1 $dataset`
for vesselid in $(tail +2 $dataset | cut -d, -f2 | sort -u); do echo $header >
${vesselid}.csv; done
tail +2 $dataset | awk -F, '{ fname=$2".csv"; print >> fname; close(fname) }'
```

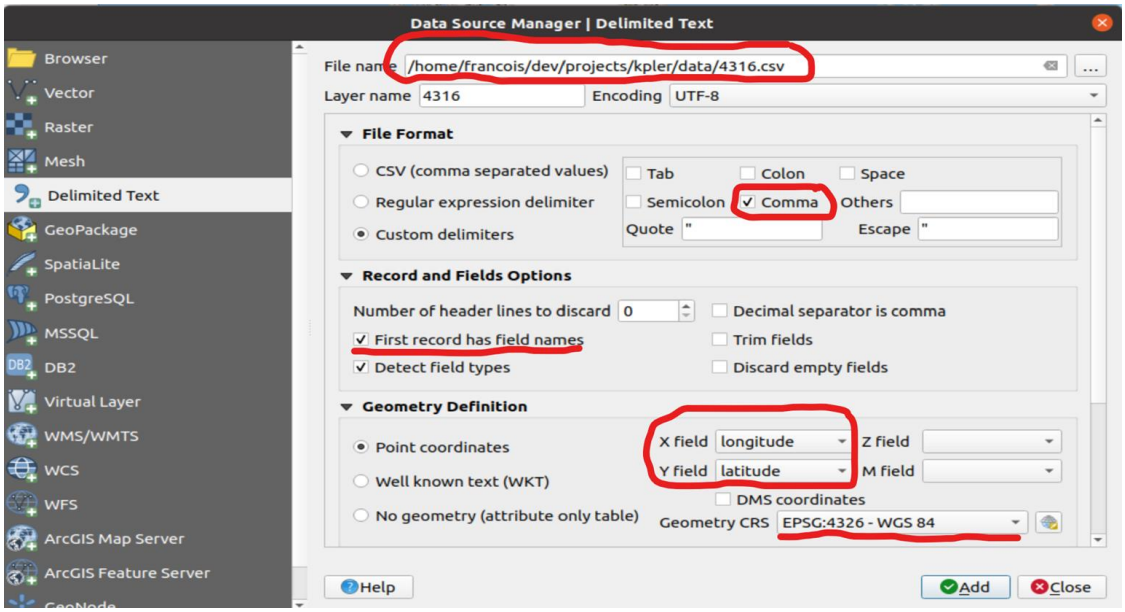
1.3.3 Exploring the dataset using QGIS

It is useful to have the HCMGIS plugin installed in QGIS so that we can display the vessel tracks over a nice map.

Start your GQIS and create a new project.

Then in the HCMGIS menu click Base Map > Google Terrain Hybrid.

To add vessel tracks: menu Layer > Add Layer > Add delimited text layer



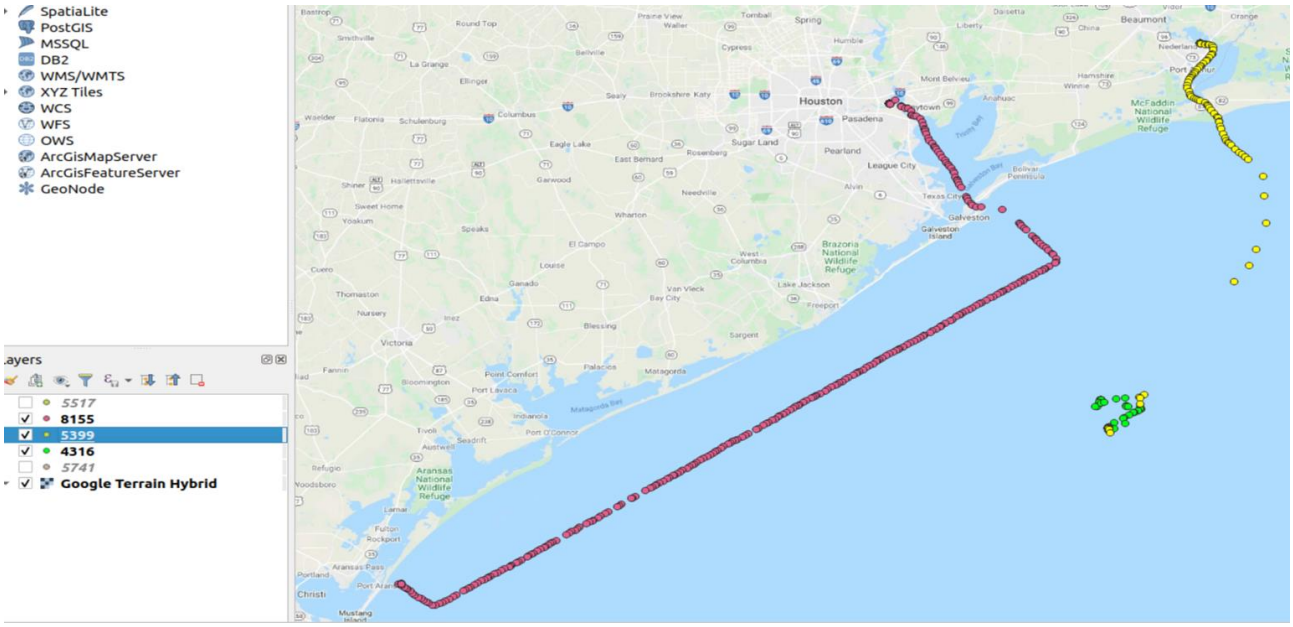
Add all the vessel tracks then show/hide them using the tick box that appears on the left menu in the Layers list.

We can already see that there are some interesting vessel tracks like 4316, 4965, 5353, 5399, 5353, 5517, 5741, 7002, 8155.

For example the vessel 8155 plotted in **red** below shows an interesting track.

In **yellow** we see vessel 5399 moving from the sea to a port.

In **green** we see vessel 4316 staying away from any port and being close to vessel 5399 movements, this could indicate an STS operation, **but it is hard to say without including any event time information, distance between vessels, speed**, ...



1.3.4 Basic algorithm to identify STS events

Based on the informations we got from the videos and articles described upper, we know that:

- STS operations are conducted at a **limited speed**, between 0.0 to 6 knots, sometimes the vessels are even just drifting
- STS operations are conducted between two vessels that must have a very similar **heading** and **speed**
- two vessels undergoing STS operations must be **very close to each other**, knowing that supertankers can be up to around 70 meters large. Latitude and longitude will be useful here.
- weather conditions could allow us to know if STS events are possible, but this is not something we have access to in this challenge.
- the **area** where the vessels are located could be a parameter to allow to include/reject AIS events during STS events detection (if an AIS event is not inside a defined area, then reject it); supertankers cannot come too close to the sea shore, and in Galveston area it seems that STS events happen between 20 and 60 nautical miles for example. We won't use this as a parameter for this challenge due to lack of time.
- the **event time** will be very important to identify if two events meeting the conditions described previously happen at approximatively the same time. Unfortunately AIS events are not sent in a consistent manner, so this will impact our capacity to accurately detect STS events. Also remember that STS events last between 1 or 2 hours up to a day.

Based on all this factors, we need to be able to detect AIS events appearing in a time window

that is not too large, so that we don't detect vessels at the same position but at very different time, and those AIS events should meet the conditions described before, knowing that the time window should shift while overlapping with the previous one. We have to use shifting (or hopping) windows because if an STS event is made of one AIS event located at the end of Window A and the second event is located at the beginning of Window B then we won't be able to match them together.

A simplified algorithm could be :

1 collect all AIS events (E) in a time window (T)

2 initialize an empty **store** of STS events

3 compare the combinations of all individual event of (E), by **pair** of events

  3.1 do the 2 events meet the following conditions ?

    3.1.1 the **speed** for this 2 events is similar (+- 5 knots)

    3.1.2 the **heading** for this 2 events is similar (+- 5 degrees)

    3.1.3 the **distance** between the vessels of this 2 events is below 100 meters

    3.1.4 the vessel speed of this 2 events is below 6 knots

  3.2 if Yes then generate an STS event holding this 2 events and add it to the store

  3.3 is there other events to compare ?

  3.4 if Yes then move to the next pair of events and loop to (3.1)

  3.5 export the store of STS events

4 move the window (T) forward by (N) minutes

5 go to (1)

For this algorithm the time window (T) should not be too large. Knowing that vessels do not move very quickly a good value for (T) could be 10 minutes.

Note that there will also be duplicates of STS events due to the window mechanism so we will have to deduplicate them.

1.3.5 Implementation

The provided repository contains an implementation using Scala and Kafka Streams so that we can use the hopping windows mechanism offered by Kafka.

In terms of output we have a variety of choice and it all depends on the consumer requirements :

1. we could produce tuples of STS events that would correspond to the original AIS events
2. we could produce an object per STS event detected, each object would contain a list of AIS events

To simplify the work I choosed the first option, to produce tuples of AIS events that I convert to STS events.

Using Kafka Streams means having partitions and one to many consumers, to leverage this process we will introduce the concept of a HaborId and we will have the HarborId as a key for our records, and all the events related to this HarborId will be streamed with this HarbordId in Kafka.

Here are some of the object declarations we are using :

An AIS event is defined like below :

```
case class AisEvent(harborId: HarborId,
                    eventId: EventId,
                    vesselId: VesselId,
                    latitude: Latitude,
                    longitude: Longitude,
                    eventTime: EventTime,
                    speed: Speed,
                    course: Course,
                    heading: Heading,
                    draught: Draught,
                    providerId: Long,
                    aisType: String,
                    point: String,
                    newNavigationalStatus: String)
```

An STS object would be defined like below :

```
case class Sts(harborId: HarborId,
               eventId: EventId,
               vesselId: VesselId,
               latitude: Latitude,
               longitude: Longitude,
               eventTime: EventTime,
               speed: Speed,
               heading: Heading,
               course: Course)
```

An STS event object would be defined like below :

```
case class StsEvent(sts1: Sts, sts2: Sts)
```

1.4 Running the project

You will need a running Kafka and follow the steps described in the README.md file provided in the zip file for this challenge, then just run the project through IntelliJ.

The README.md file describes how to inject the Galveston.csv file in the ais input topic, and the module produces its output in the sts output topic and displays each individual STS event on the screen, the output is the following :

I have highlighted the vessels for the 3 different STS events that the application has detected :

- Vessels 6653 and 7642
- Vessels 4316 and 5399
- Vessels 4721 and 5030

k=StsEventKey(120045806421,232621445,232622287) v=StsEvent(<<vessel=6653 sp=0.8 hd=22 co=5.0 eid=232621445 ets=2017-11-26 06:30:12.0>>,<<vessel=7642 sp=0.7 hd=20 co=3.0 eid=232622287 ets=2017-11-26 06:30:37.0>>)

k=StsEventKey(120045806421,232621905,232622010) v=StsEvent(<<vessel=4316 sp=0.1 hd=137 co=239.0 eid=232621905 ets=2017-11-26 06:29:07.0>>,<<vessel=5399 sp=0.3 hd=142 co=252.0 eid=232622010 ets=2017-11-26 06:25:52.0>>)

k=StsEventKey(120045806421,232629248,232630114) v=StsEvent(<<vessel=6653 sp=0.1 hd=18 co=249.0 eid=232629248 ets=2017-11-26 07:00:01.0>>,<<vessel=7642 sp=0.2 hd=16 co=267.0 eid=232630114 ets=2017-11-26 06:59:58.0>>)

k=StsEventKey(120045806421,232649602,232649435) v=StsEvent(<<vessel=4721 sp=3.6 hd=274 co=265.0 eid=232649602 ets=2017-11-26 08:30:40.0>>,<<vessel=5030 sp=3.6 hd=271 co=265.0 eid=232649435 ets=2017-11-26 08:29:53.0>>)

k=StsEventKey(120045806421,232693001,232693822) v=StsEvent(<<vessel=6653 sp=0.0 hd=65 co=313.0 eid=232693001 ets=2017-11-26 11:30:22.0>>,<<vessel=7642 sp=0.0 hd=64 co=187.0 eid=232693822 ets=2017-11-26 11:29:59.0>>)

k=StsEventKey(120045806421,232699020,232699779) v=StsEvent(<<vessel=6653 sp=0.0 hd=65 co=227.0 eid=232699020 ets=2017-11-26 11:50:12.0>>,<<vessel=7642 sp=0.1 hd=64 co=331.0 eid=232699779 ets=2017-11-26 11:57:00.0>>)

k=StsEventKey(120045806421,232705552,232706351) v=StsEvent(<<vessel=6653 sp=0.0 hd=68 co=188.0 eid=232705552 ets=2017-11-26 12:20:52.0>>,<<vessel=7642 sp=0.0 hd=66 co=250.0 eid=232706351 ets=2017-11-26 12:14:58.0>>)

k=StsEventKey(120045806421,232711980,232712809) v=StsEvent(<<vessel=6653 sp=0.0 hd=60 co=199.0 eid=232711980 ets=2017-11-26 12:59:20.0>>,<<vessel=7642 sp=0.1 hd=59 co=190.0 eid=232712809 ets=2017-11-26 12:56:59.0>>)

k=StsEventKey(120045806421,232717579,232718382) v=StsEvent(<<vessel=6653 sp=0.1 hd=56 co=201.0 eid=232717579 ets=2017-11-26 13:25:33.0>>,<<vessel=7642 sp=0.0 hd=55 co=161.0 eid=232718382 ets=2017-11-26 13:23:58.0>>)

k=StsEventKey(120045806421,232724936,232725739) v=StsEvent(<<vessel=6653 sp=0.1 hd=47 co=200.0 eid=232724936 ets=2017-11-26 13:55:40.0>>,<<vessel=7642 sp=0.1 hd=48 co=317.0 eid=232725739 ets=2017-11-26 13:47:59.0>>)

k=StsEventKey(120045806421,232730631,232731435) v=StsEvent(<<vessel=6653 sp=0.0 hd=48 co=170.0 eid=232730631 ets=2017-11-26 14:25:51.0>>,<<vessel=7642 sp=0.1 hd=45 co=209.0 eid=232731435 ets=2017-11-26 14:20:59.0>>)

k=StsEventKey(120045806421,232737072,232737876) v=StsEvent(<<vessel=6653 sp=0.0 hd=47 co=137.0 eid=232737072 ets=2017-11-26 14:55:11.0>>,<<vessel=7642 sp=0.1 hd=42 co=159.0 eid=232737876 ets=2017-11-26 14:47:59.0>>)

k=StsEventKey(120045806421,232750638,232751406) v=StsEvent(<<vessel=6653 sp=0.1 hd=47 co=173.0 eid=232750638 ets=2017-11-26 15:56:12.0>>,<<vessel=7642 sp=0.1 hd=45 co=172.0 eid=232751406 ets=2017-11-26 15:53:58.0>>)

k=StsEventKey(120045806421,232759439,232760224) v=StsEvent(<<vessel=6653 sp=0.1 hd=46 co=142.0 eid=232759439 ets=2017-11-26 16:25:31.0>>,<<vessel=7642 sp=0.0 hd=43 co=130.0 eid=232760224 ets=2017-11-26 16:23:59.0>>)

k=StsEventKey(120045806421,232765327,232766129) v=StsEvent(<<vessel=6653 sp=0.1 hd=50 co=145.0 eid=232765327 ets=2017-11-26 16:54:50.0>>,<<vessel=7642 sp=0.0 hd=48 co=323.0 eid=232766129 ets=2017-11-26 16:53:59.0>>)

k=StsEventKey(120045806421,232771969,232772760) v=StsEvent(<<vessel=6653 sp=0.1 hd=56 co=129.0 eid=232771969 ets=2017-11-26 17:25:32.0>>,<<vessel=7642 sp=0.1 hd=54 co=28.0 eid=232772760 ets=2017-11-26 17:24:00.0>>)

k=StsEventKey(120045806421,232777461,232778258) v=StsEvent(<<vessel=6653 sp=0.0 hd=47 co=158.0 eid=232777461 ets=2017-11-26 18:01:12.0>>,<<vessel=7642 sp=0.1 hd=47 co=151.0 eid=232778258 ets=2017-11-26 17:59:59.0>>)

k=StsEventKey(120045806421,232784297,232785076) v=StsEvent(<<vessel=6653 sp=0.1 hd=49 co=177.0 eid=232784297 ets=2017-11-26 18:25:21.0>>,<<vessel=7642 sp=0.0 hd=47 co=171.0 eid=232785076 ets=2017-11-26 18:23:59.0>>)

k=StsEventKey(120045806421,232790032,232790838) v=StsEvent(<<vessel=6653 sp=0.0 hd=50 co=160.0 eid=232790032 ets=2017-11-26 18:56:02.0>>,<<vessel=7642 sp=0.0 hd=48 co=138.0 eid=232790838 ets=2017-11-26 18:54:00.0>>)

k=StsEventKey(120045806421,232794437,232795255) v=StsEvent(<<vessel=6653 sp=0.1 hd=53 co=142.0 eid=232794437 ets=2017-11-26 19:26:21.0>>,<<vessel=7642 sp=0.1 hd=53 co=143.0 eid=232795255 ets=2017-11-26 19:23:59.0>>)

k=StsEventKey(120045806421,232802403,232803196) v=StsEvent(<<vessel=6653 sp=0.1 hd=51 co=168.0 eid=232802403 ets=2017-11-26 19:56:21.0>>,<<vessel=7642 sp=0.1 hd=51 co=163.0 eid=232803196 ets=2017-11-26 19:50:58.0>>)

k=StsEventKey(120045806421,232811012,232811819) v=StsEvent(<<vessel=6653 sp=0.1 hd=52 co=6.0 eid=232811012 ets=2017-11-26 20:23:52.0>>,<<vessel=7642 sp=0.1 hd=50 co=37.0 eid=232811819 ets=2017-11-26 20:14:59.0>>)

k=StsEventKey(120045806421,232816137,232816901) v=StsEvent(<<vessel=6653 sp=0.1 hd=51 co=155.0 eid=232816137 ets=2017-11-26 20:55:51.0>>,<<vessel=7642 sp=0.0 hd=48 co=27.0 eid=232816901 ets=2017-11-26 20:53:59.0>>)

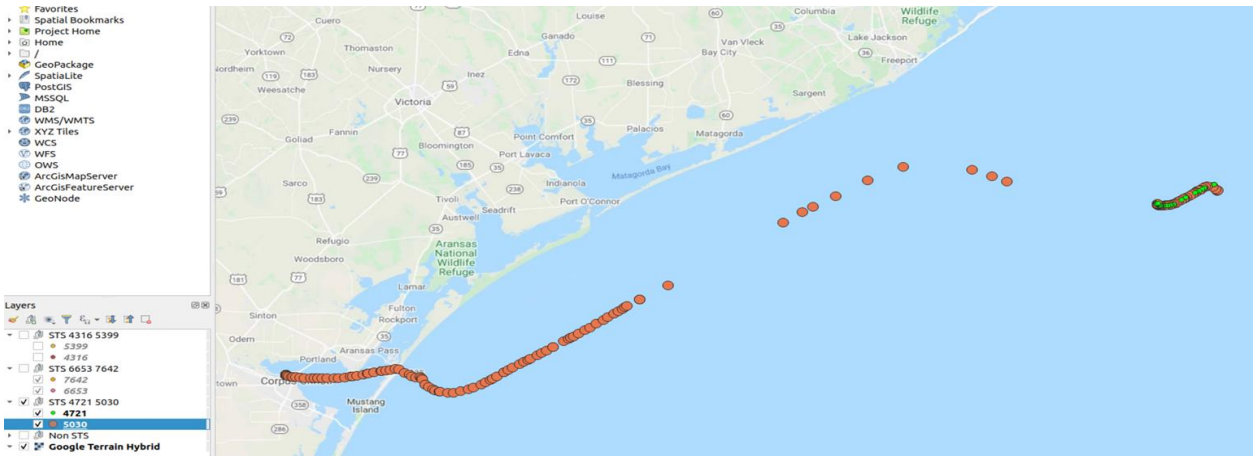
k=StsEventKey(120045806421,232822547,232823356) v=StsEvent(<<vessel=6653 sp=0.0 hd=46 co=148.0 eid=232822547 ets=2017-11-26 21:26:21.0>>,<<vessel=7642 sp=0.0 hd=44 co=165.0 eid=232823356 ets=2017-11-26 21:23:59.0>>)

1.4.1 Plotting the STS events with QGIS

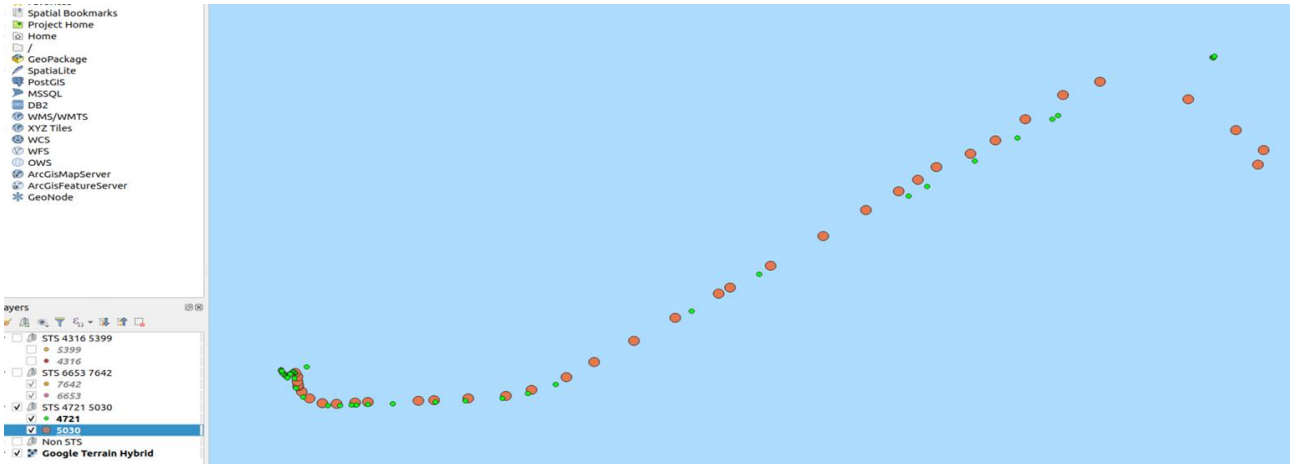
Please note that the QGIS representations below do not show only STS events, they show all the movements for the pair of vessels for which an STS event have been detected. So even if sometimes it can be confusing, it is also very useful to see how some vessels move from the sea shore farther in the sea to meet supertankers and conduct their STS operations.

1.4.2 Vessels 4721 and 5030

The QGIS representation of the tracks for this 2 vessels clearly shows an STS event, we can confirm this by reading the values provided in the module output (event time, positions of the vessels, speed, heading). Note how 5030 moves close to the city of Corpus Christi.

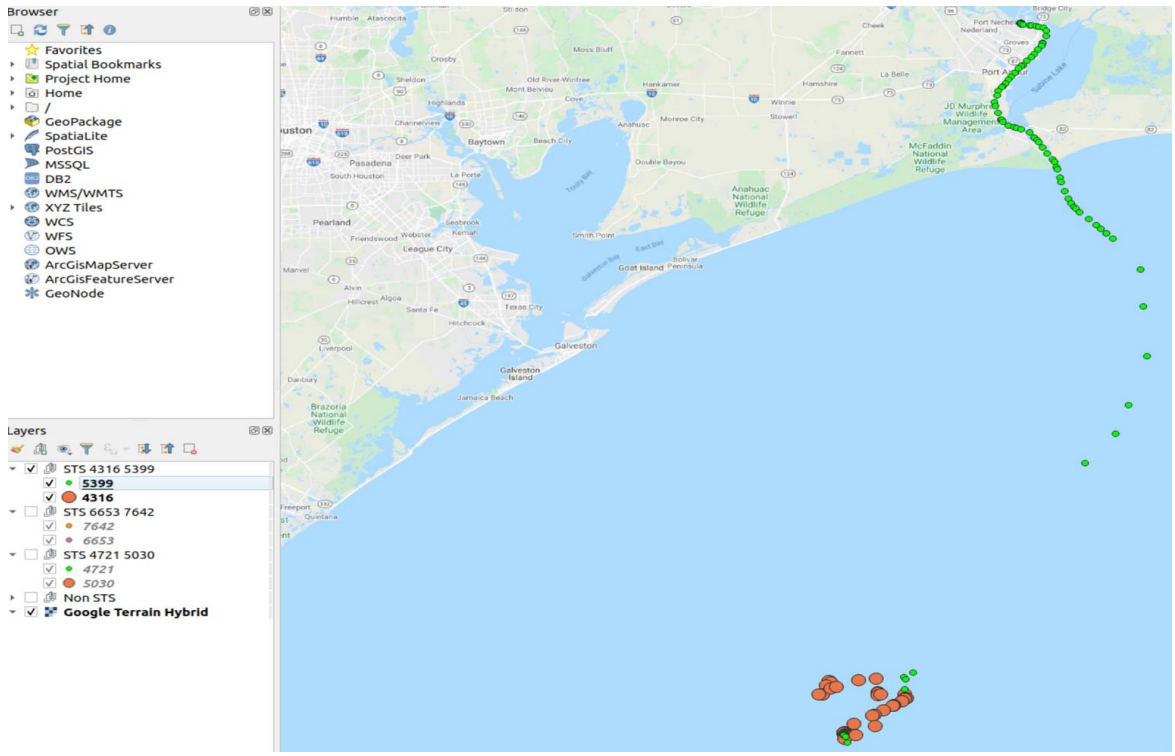


Zooming on the right part of the map where the STS happens shows clearer details :

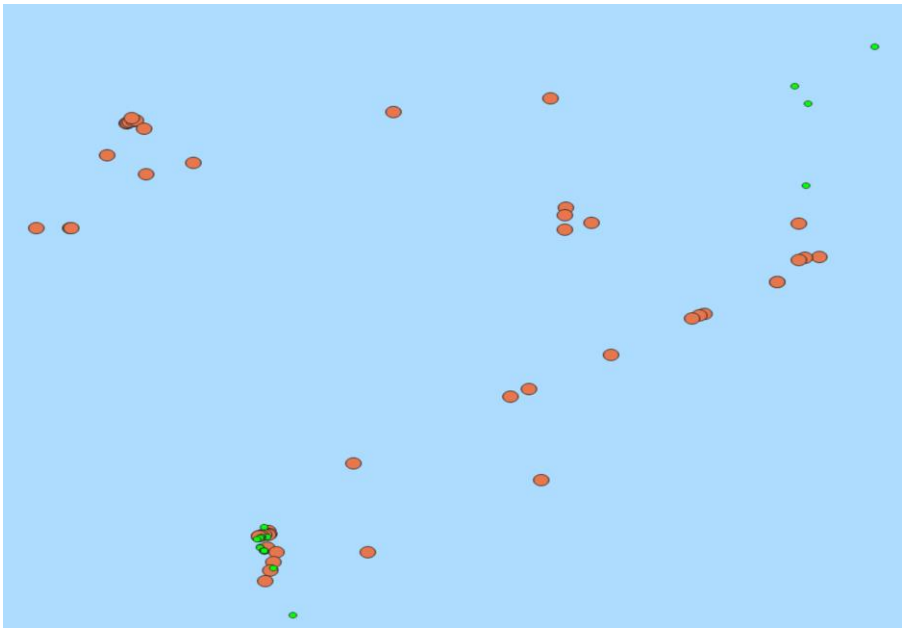


1.4.3 Vessels 4316 and 5399

The QGIS representation of the tracks for this 2 vessels shows an STS event, we can confirm this by reading the values provided in the module output (event time, positions of the vessels, speed, heading). It is intesting to see how 5399 moves in and out.

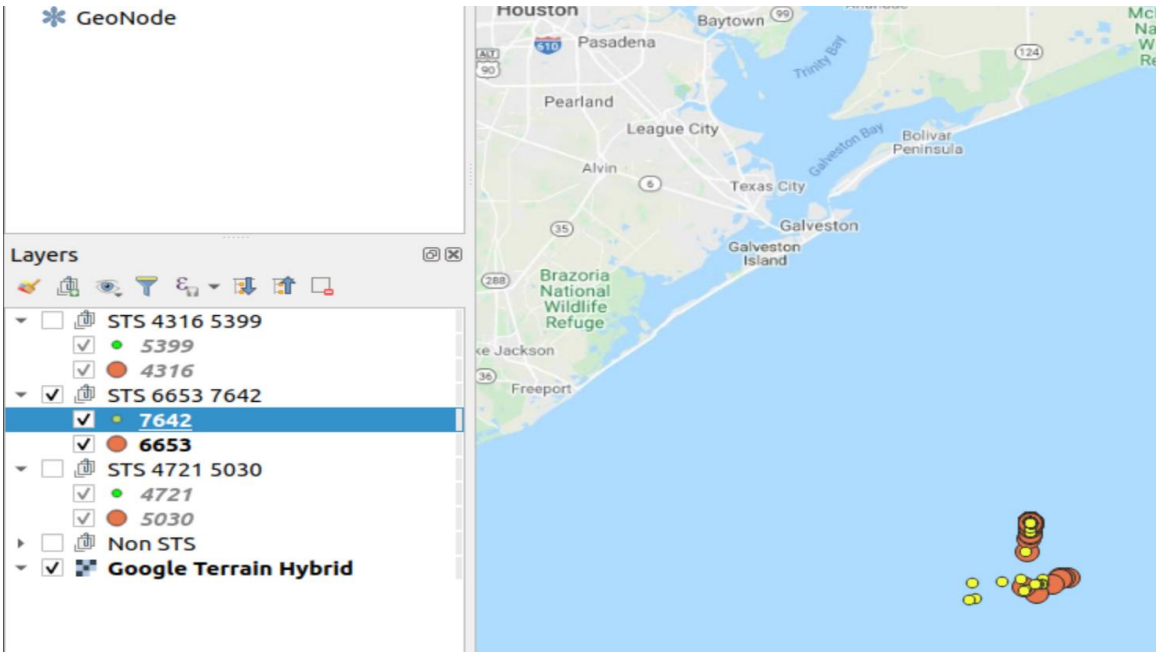


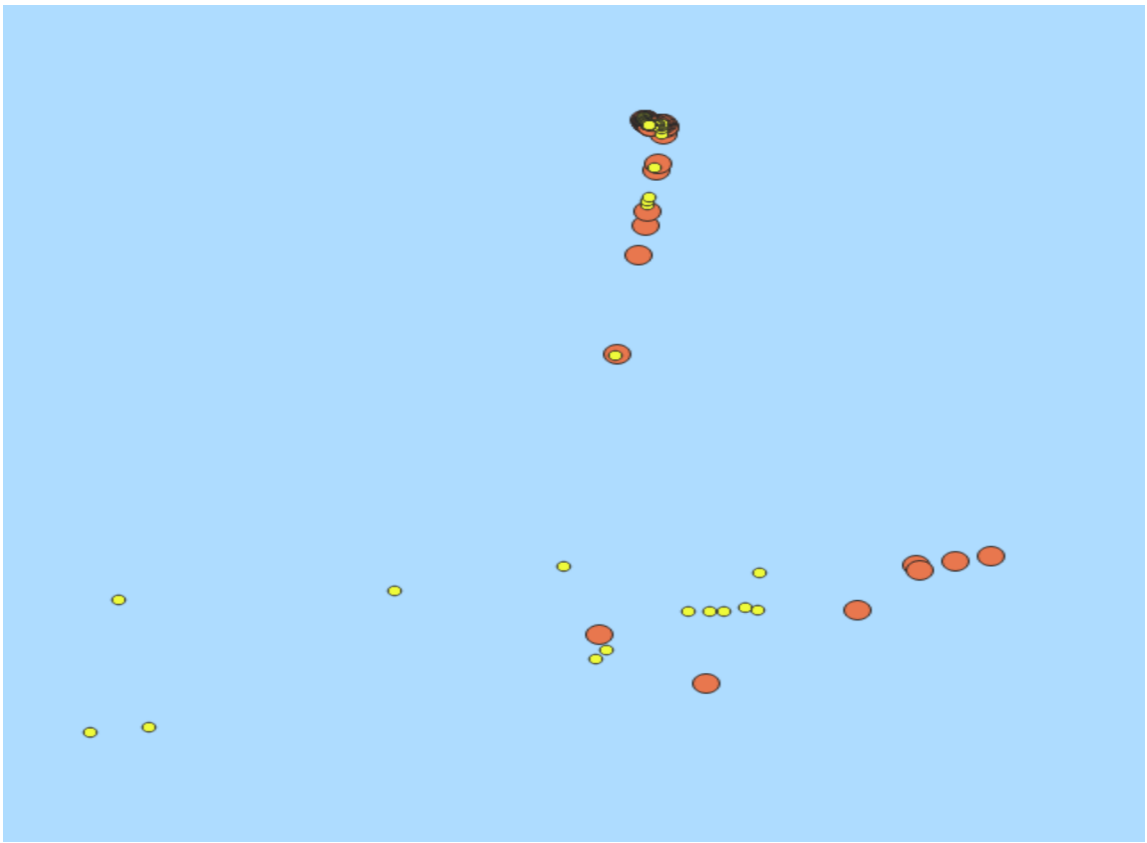
Zomming on the left part of the map where the STS happens shows clearer details but also that AIS events are not sent consistently.



1.4.4 Vessels 6653 and 7642

The QGIS representation of the tracks for this 2 vessels shows an STS event, we can confirm this by reading the values provided in the module output (event time, positions of the vessels, speed, heading). It is not as clear as the other tracks as we don't see a daughter vessel going back to a port.





2 Oil Product Storage Pipeline Architecture

For this problem we have to build a pipeline that allows to visualize the variations of oil products in petrol tanks over time, based on satellite images, for a list of sites (starting with 10 sites, up to 300 sites). We won't describe the algorithm to process the satellite images and compute the level of product based on the roof shadows. We have to extrapolate the volumes when we are missing satellite images. Satellite images are available through a REST API.

We can imagine the following architecture:

1 we have a Query Service (QS) that polls for satellite images multiple times a day (we may not have only one API to call, so we have multiple instances of our QS)

2 when a new satellite image is available QS stores it on an S3 bucket (depending on agreement with the image provider)

3 QS informs our image processing service (IPS) that a new image is available with all the details about where it is stored, etc

3.1 the IPS computes the oil volume for each petrol tank that is shown on this image, (identifying tanks, distributing the computation of individual tanks over multiple instances)

3.2 the IPS computes extrapolated oil volume for all missing days between the previous image and the current image for each petrol tank he has just computed an oil volume

3.3 the IPS stores the computed oil volumes and extrapolated oil volumes (with a discriminator allowing to identify which is real and which is extrapolated, for which site and petrol tank is the data for, etc) in a database and sends an event in a queue (OILVOL) for the aggregation service (AS)

3.4 An aggregation service reads new events from OILVOL and produces views for different visualization services, this services may eventually subscribe to the aggregation service.

3.5 A REST API exposes both the raw Oil Volumes and the views for a front end to visualize it.

2.1 Oil Volumes

The minimum data we could store for oil volumes is the following :

- site id
- petrol tank id
- image id (can be null for extrapolated data)
- oil volume
- volume timestamp (date when it is computed for)
- oil volume type (real, extrapolated)
- computed_at

Of course we would have a table for :

- sites
  - site name
  - site owner
  - coordinates (lat, long)
  - …
- tanks per site

- site id
- coordinate of the petrol tank (lat, long)
- type of tank (petrol, ...)
- size of tank (to allow for oil volume computation)

Below is a diagram showing a simplified version of our pipeline :

- we could use Kafka as a message bus, Postgres to store Oil Volumes, and Elasticsearch to store our views.
- we would have multiple instances of the Query Service and the Image Processing Service, so that we can process things in parallel.
- we could serve our REST APIs with Golang or Scala/Play framework.