Processing real time AIS messages and integrating with third party geo data using Sesam

Tom Bech

Sesam product developer
tom.bech@sesam.io

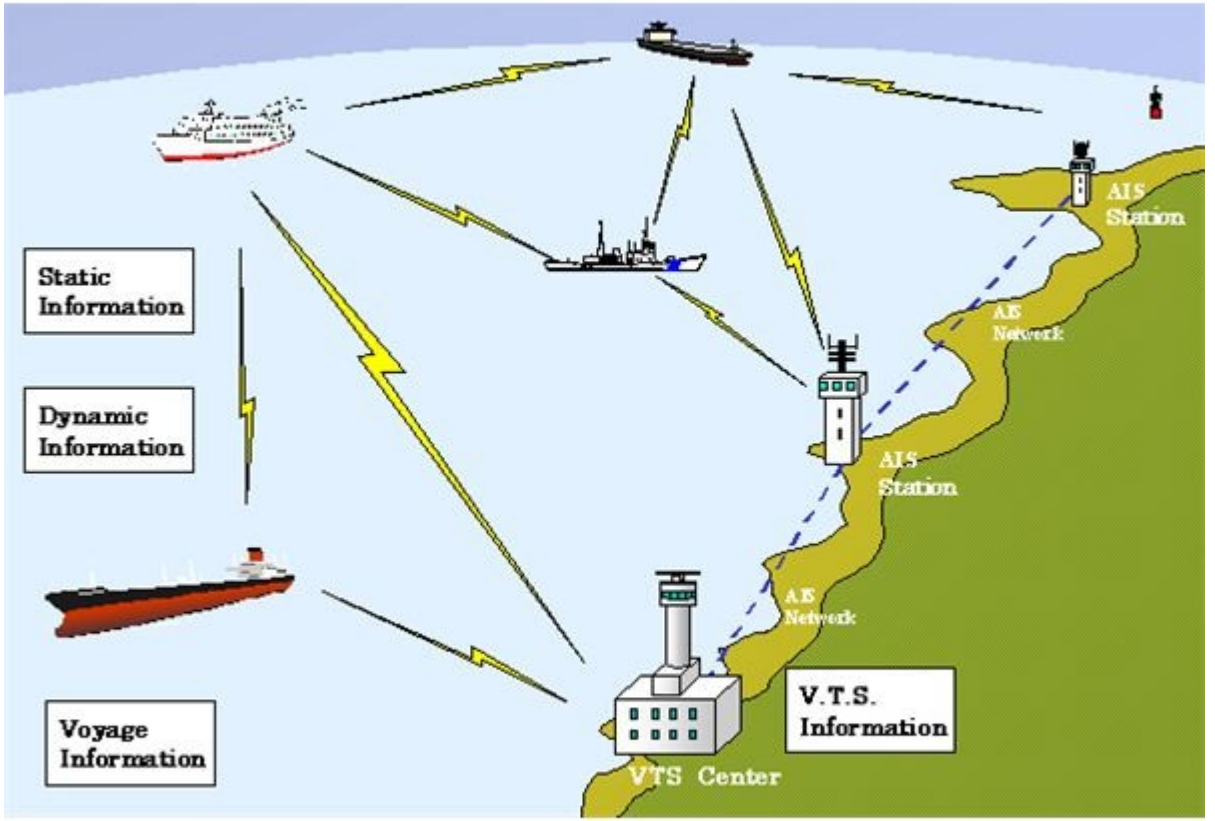
Background & motivation

- Read interesting Norwegian Digi article about Gov't opening up "AIS" real time ship traffic data to the general public
- Disappointed to learn data not really open, article turn out to be about a web map *applicaton*
- Public *data* only available as socket with binary "raw" data (full access to "proper data" still subject to application & approval)

AIS 101

- AIS = ***A***utomatic ***I***dentification ***S***ystem
- UN standard (IEC 62320-1)
- Used world-wide for maritime traffic

- Real time messages from ships
  - ID, name and callsigns
  - Position updates, speed, bearing, status
  - Emergencies
  - Cargo information & destination
  - Metadata; type, size



AIS con’t

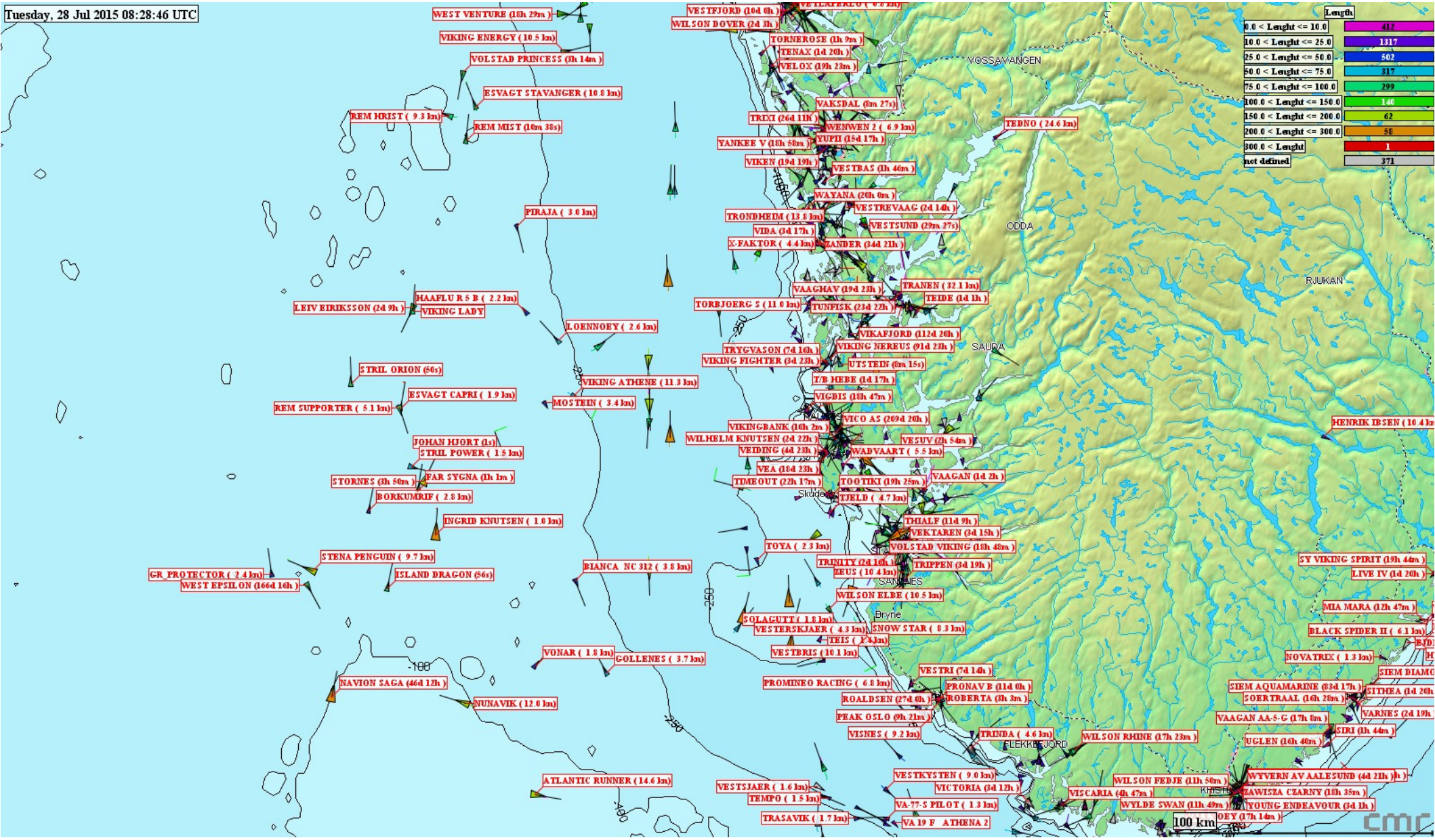
- Messages broadcast, rebroadcast
- Over-the-horizon media: VHF/UHF, satellites
  - Radar limited to visible range
- Picked up by transponders
  - Stations on land - i.e. basestations
  - Satelittes
- All larger vessels in world have AIS equipment
- Many leisure boats as well

AIS in Norway

- “Kystverket” owns and operates
  - 50 base stations along the coast
  - Two satellites in polar orbit

- Infrastructure to process and analyze AIS data
  - Rebroadcasts AIS messages on the internet
  - ~30-100 messages/sec


Friday, 08 May 2009 06:42:23 UTC
Ship type
Cargo ship 3606
Tanker 1508
Vessel 958
Passenger ship 955
not defined 927
Tug boat 862
Other 707
Fishing Vessel 691
SAR vessel 255
Pleasure craft 248
Pilot vessel 213
Various 399
2573525
2573485
2573475
2573515
2573505
HARSTAD
2573465
2573455
NARVIK
2573445
2573435
2573425
100 km
cmr


Tuesday, 28 Jul 2015 08:28:46 UTC
WEST VENTURE (18h 29m)
VIKING ENERGY (10.5 kn)
VOLSTAD PRINCESS (8h 14m)
ESVAGT STAVANGER (10.8 kn)
REM HRIST (9.3 kn)
REM MIST (10m 38s)
PIRAJA (3.0 kn)
HAAFLU R S B (2.2 kn)
LEIV EIRIKSSON (2d 9h)
VIKING LADY
LOENNOEY (2.6 kn)
STRIL ORION (60s)
ESVAGT CAPRI (1.9 kn)
REM SUPPORTER (5.1 kn)
VIKING ATHENE (11.3 kn)
MOSTEIN (3.4 kn)
JOHAN HJORT (1s)
STRIL POWER (1.5 kn)
STORNES (3h 50m)
FAR SYGNA (1h 1m)
BORKUMRIF (2.8 kn)
INGRID KNUTSEN (1.0 kn)
STENA PENGUIN (9.7 kn)
GR_PROTECTOR (2.4 kn)
WEST EPSILON (166d 16h)
ISLAND DRAGON (56s)
BIANCA NC 312 (3.8 kn)
VONAR (1.8 kn)
GOLLENES (3.7 kn)
NAVION SAGA (46d 12h)
NUNAVIK (12.0 kn)
ATLANTIC RUNNER (14.6 kn)
VESTFJORD (10d 0h)
WILSON DOVER (2d 3h)
TORNEROSE (1h 9m)
TENAX (1d 20h)
VELOX (19h 23m)
VOSSAVANGEN
VAKSDAL (8m 27s)
TRIXI (26d 11h)
WENWEN 2 (6.9 kn)
TEDNO (24.6 kn)
YANKEE V (18h 58m)
YUPII (15d 17h)
VIKEN (19d 19h)
VESTBAS (1h 46m)
WAYANA (20h 0m)
VESTREVAAG (2d 14h)
TRONDHEIM (13.8 kn)
VESTSUND (29m 27s)
ODDA
VIDA (3d 17h)
X-FAKTOR (4.4 kn)
ZANDER (34d 21h)
VAAGHAV (19d 23h)
TRANEN (32.1 kn)
TORBJOERG S (11.0 kn)
TEIDE (1d 1h)
TUNFISK (23d 22h)
RJUKAN
VIKAFJORD (112d 20h)
TRYGVASON (7d 16h)
VIKING NEREUS (91d 23h)
SAURA
VIKING FIGHTER (3d 23h)
UTSTEIN (8m 15s)
T/B HEBE (1d 17h)
VIGDIS (18h 47m)
VICO AS (209d 20h)
VIKINGBANK (10h 2m)
HENRIK IBSEN (10.4 kn)
WILHELM KNUTSEN (2d 22h)
VESUV (2h 54m)
VEIDING (4d 23h)
WADVAART (5.5 kn)
VEA (18d 23h)
TIMEOUT (22h 17m)
TOOTIKI (19h 25m)
VAAGAN (1d 2h)
Skuld
TJELD (4.7 kn)
THALF (11d 9h)
VEKTAREN (3d 15h)
TOYA (2.3 kn)
VOLSTAD VIKING (18h 48m)
TRINITY (2d 16h)
TRIPPEN (3d 19h)
ZEUS (10.4 kn)
SY VIKING SPIRIT (19h 44m)
LIVE IV (1d 20h)
SANDNES
WILSON ELBE (10.5 kn)
MIA MARA (12h 47m)
SOLAGUTT (1.8 kn)
Bryne
VESTERSKJAER (4.3 kn)
SNOW STAR (8.3 kn)
BLACK SPIDER II (6.1 kn)
TEIS (7.3 kn)
VESTBRIS (10.1 kn)
NOVATRIX (1.3 kn)
VESTRI (7d 14h)
PROMINEO RACING (6.8 kn)
PRONAV B (11d 0h)
SIEM AQUAMARINE (83d 17h)
ROBERTA (3h 3m)
SOERTRAAL (16h 28m)
ROALDSEN (27d 0h)
PEAK OSLO (9h 21m)
VAAGAN AA-5-G (17h 8m)
VISNES (9.2 kn)
TRINDA (4.6 kn)
WILSON RHINE (17h 23m)
FLEKKEFJORD
UGLEN (16h 40m)
VESTKYSTEN (9.0 kn)
WILSON FEDJE (11h 50m)
VESTSJAER (1.6 kn)
VICTORIA (3d 12h)
VISCARIA (4h 47m)
TEMPO (1.5 kn)
VA-77-S PILOT (1.3 kn)
WYLDE SWAN (11h 49m)
TRASAVIK (1.7 kn)
VA 19 F ATHENA 2
100 km
Length
0.0 < Lenght <= 10.0 412
10.0 < Lenght <= 25.0 1317
25.0 < Lenght <= 50.0 502
50.0 < Lenght <= 75.0 317
75.0 < Lenght <= 100.0 299
100.0 < Lenght <= 150.0 140
150.0 < Lenght <= 200.0 62
200.0 < Lenght <= 300.0 58
300.0 < Lenght 1
not defined 371
cmr

The Challenge!

- Want to use AIS data + a search engine to do geo-queries like:
  - Which ships are near “Bergen”?
  - How many cargo ships are in Norwegian waters today?
  - Which ships are nearest vs another ship?
  - How many fishing boats are active near Lofoten at the moment?

- Additional challenge: from scratch - and in one day!

Programmer: Ooh. Very cool task, but pretty ambitious! Can I use Sesam as the integration platform?

Boss: Sure, use whatever tools you think is best for the job

Challenge accepted!

What is Sesam?

- Sesam is an integration platform
- Employs a Data Oriented Architecture (DOA) to make integration more manageable, scalable, powerful, expressive and agile
- Everything is data and everything is explicit
  - Including configuration, logging etc
- A datahub as a central component
  - Schema-less data store that consumes or is fed the data from the systems that are to be integrated
  - Solves the “cache-invalidation” problem in integration context
  - Push&pull source/transformation/sink + microservices = Pipe
  - Pipe+Pipe=Flow
  - Extensive REST API + command line client + GUI

So..

1) Learn/research about AIS
2) Consume AIS data into Sesam somehow - write microservice?
3) Integrate AIS data with third party data (geolocated places) and compute nearest - another microservice?
4) Learn/research Elasticsearch
5) Insert result and changes into Elasticsearch
6) Figure how to do the queries
7) Profit!

..

Starting point

- http://www.kystverket.no/Nyheter/2016/september/apner-ais-for-publikum
- A subpage mentions a AIS socket (IP+port)
- Tested it in Firefox. Got:

```
!BSVDM,1,1,,B,H3m<Od4N>F34u5K<=ojjn00H8220,0*73
!BSVDM,1,1,,A,35UeSP5000Puj;>V<B`;02mV0000,0*0F
!BSVDM,1,1,,A,13mM0u00001FlEdWnhUuL:OT0@NW,0*17
!BSVDM,2,1,1,A,53mN1J400000hluB2218E<=DF0T@5:1Di=@DTr0k0p?
154rdR2fLMevMeN88,0*73
!BSVDM,2,2,1,A,88888888880,2*3C
..
```

The familiar first two hours

Google

stackoverflow

GitHub

WIKIPEDIA

First round results

- An understanding of what AIS is in more detail
- Found docs for AIS spec in codemonkey-grokkable form:
  - http://catb.org/gpsd/AIVDM.html
- Found python lib for decoding AIS messages:
  - https://github.com/schwehr/libais
- Norwegian places with geo coords:
  - http://www.erikbolstad.no/geo/noreg/postnummer
- Nearest point search and geo calculus:
  - https://en.wikipedia.org/wiki/K-d_tree
  - https://en.wikipedia.org/wiki/Haversine_formula
  - https://en.wikipedia.org/wiki/Vincenty%27s_formulae

Third and fourth hour

- Forked Sesam Python Microservice template on Github + wrote Python code to read and decode AIS stream using “libais”
- The usual loop; make stupid error. Fix it. Rinse & repeat.
- Configured JSON endpoint in Sesam & posted first AIS data to Sesam

AIS MicroService Code

```
# Open socket to AIS service
s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
s.connect((options.ais_server, int(options.ais_port)))

f = s.makefile()

with requests.session() as session:
        # This loop will not end as long as the socket is open
        for msg in ais.stream.decode(f):
            message = ais.compatibility.gpsd.mangle(msg)
            message["_id"] = "%s_%s" % (message["type"], message["mmsi"])

            json_data = json.dumps(message)
            resp = session.post(url, params={}, headers={"content-type": "application/json"},
                                    data=json_data, verify=False, timeout=3600)

           resp.raise_for_status()
           resp.close()
```

Sesam receiver conf

This pipe will set up a JSON receiver so we can use HTTP POST operations at

http://sesamservice:port/api/receivers/ais_data/entitites

```
{
  "_id": "ais_data",
  "type": "pipe",
  "source": {
    "type": "http_endpoint"
  }
}
```

AIS messages


```
{
  "maneuver": 0,
  "received_stations": 29,
  "slot_timeout": 3,
  "status": 0,
  "second": 31,
  "class": "AIS",
  "scaled": True,
  "course": 0,
  "raim": True,
  "type": 1,
  "lat": 62.6781005859375,
  "spare": 0,
  "sync_state": 0,
  "device": "stdin",
  "repeat": 0,
  "lon": 6.669294834136963,
  "speed": 0,
  "accuracy": True,
  "status_text": "Under way using engine",
  "turn": NaN,
  "heading": 511,
  "mmsi": 257389600
}
```


Message type

Unique ship ID


```
{
  "type": 24,
  "device": "stdin",
  "repeat": 0,
  "shipname": "AAS KYSTSERVICE",
  "mmsi": 257389600,
  "class": "AIS",
  "scaled": true,
  "part_num": 0
}
```

Message deduplication

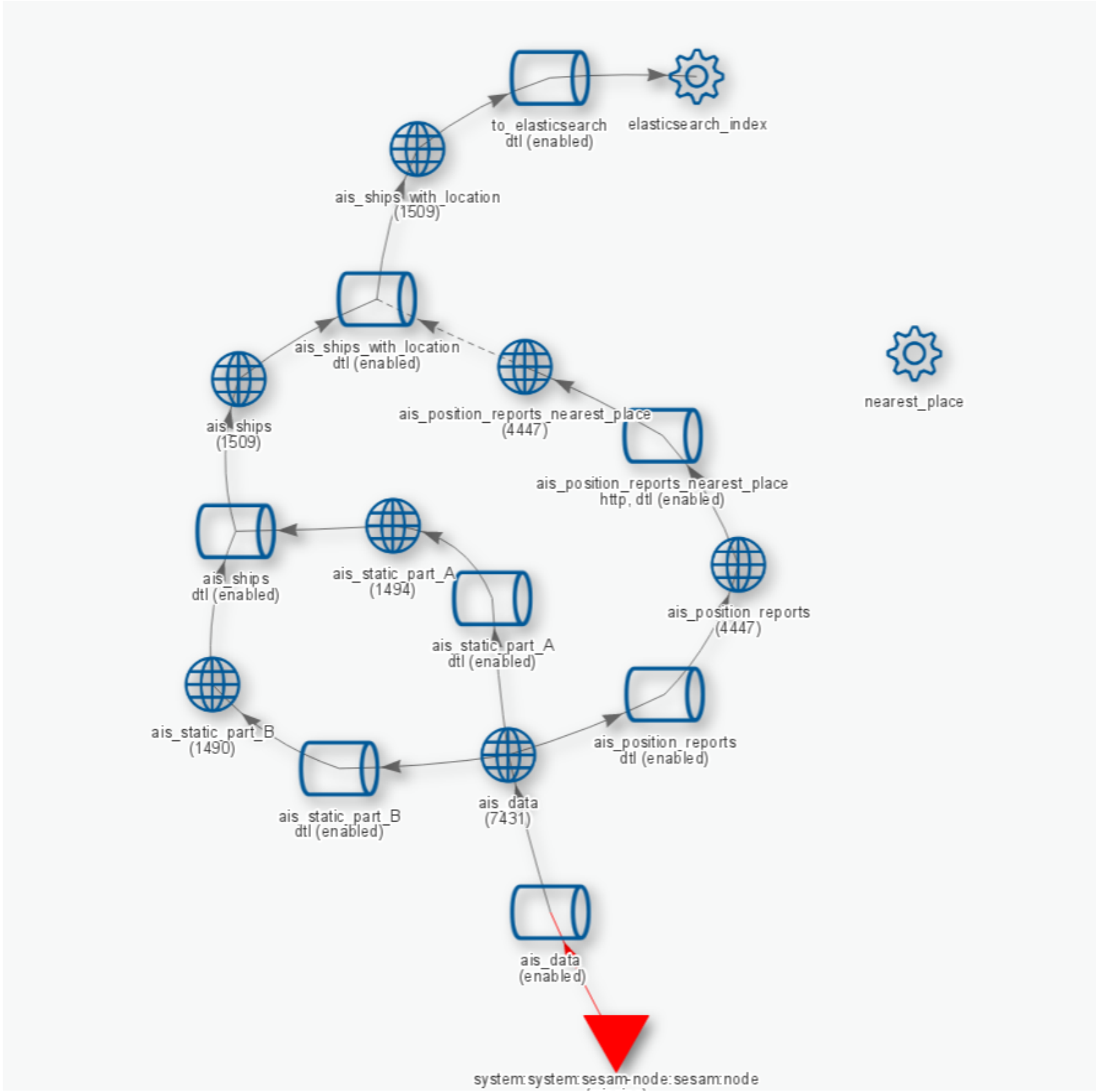
- AIS feed contain **LOTS** of duplicates (10:1)
  - Rebroadcasts + captured by multiple transponders
  - Static non-changing/repeating messages
  - Noise in data
- Sesam deduplicates automatically using hashing if given a unique entity ID
  - ID for message = MMSI + type
- Sesam keeps history of changes
  - Track ship positions over time!

Hours 4 and 5
Splitting on message type

- Split static and dynamic messages on type
- Positional updates
  - AIS types 1-3 & 18-19
- Static metadata
  - Type 5 & 24
  - These may be multipart messages!

Pipes and DTL

- Deduplicated AIS data from initial dataset piped to various new datasets based on type
  - Positional messages
  - Static data -> part A and B
  - Ship metadata from multipart messages (A U B)
  - Positional data enriched with nearest city info
  - Ship info with last known location
  - Final flow is outbound, i.e. Elasticsearch sink

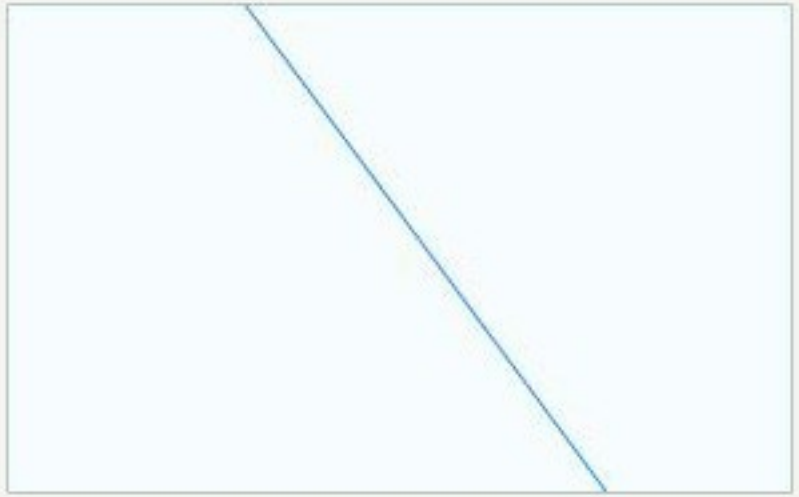

to_elasticsearch
dtl (enabled)
elasticsearch_index
ais_ships_with_location
(1509)
ais_ships_with_location
dtl (enabled)
ais_ships
(1509)
ais_position_reports_nearest_place
(4447)
nearest_place
ais_position_reports_nearest_place
http, dtl (enabled)
ais_ships
dtl (enabled)
ais_static_part_A
(1494)
ais_static_part_A
dtl (enabled)
ais_position_reports
(4447)
ais_static_part_B
(1490)
ais_position_reports
dtl (enabled)
ais_static_part_B
dtl (enabled)
ais_data
(7431)
ais_data
(enabled)
system:system:sesam-node:sesam:node

DTL transform for filtering on type

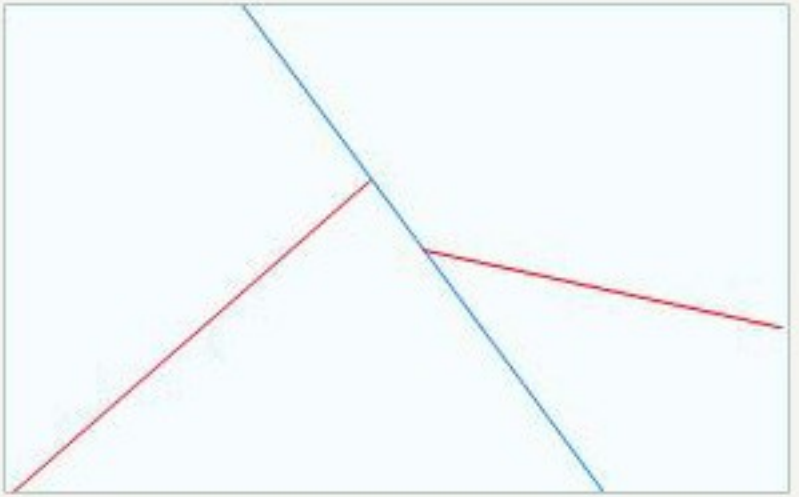
```
{
  "_id": "ais_position_reports",
  "type": "pipe",
  "source": {
  "type": "dataset",
    "dataset": "ais_data"
  },
  "transform": [
    {
        "type": "dtl",
        "name": "Filter out all but position reports (type 1-3 and 18-19)",
        "dataset": "ais_data",
        "transforms": {
            "default": [
                ["filter", ["or",
                            ["eq", "_S.type", 1],
                            ["eq", "_S.type", 2],
                            ["eq", "_S.type", 3],
                            ["eq", "_S.type", 18],
                            ["eq", "_S.type", 19]]
                ],
                ["copy", "*"]
            ]
        }
    }]
}
```

Hours 6 and 7
Nearest city microservice

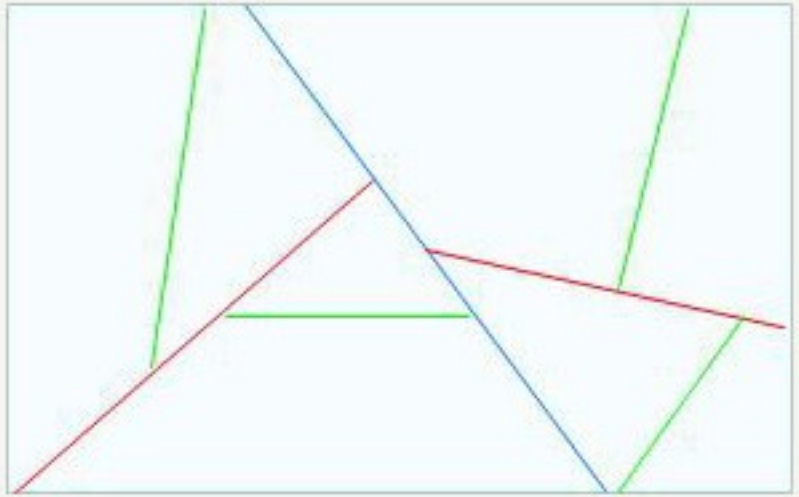
- Sesam Python HTTP transform microservice template from github
- Inserts places into a KD-tree (BSP) on startup



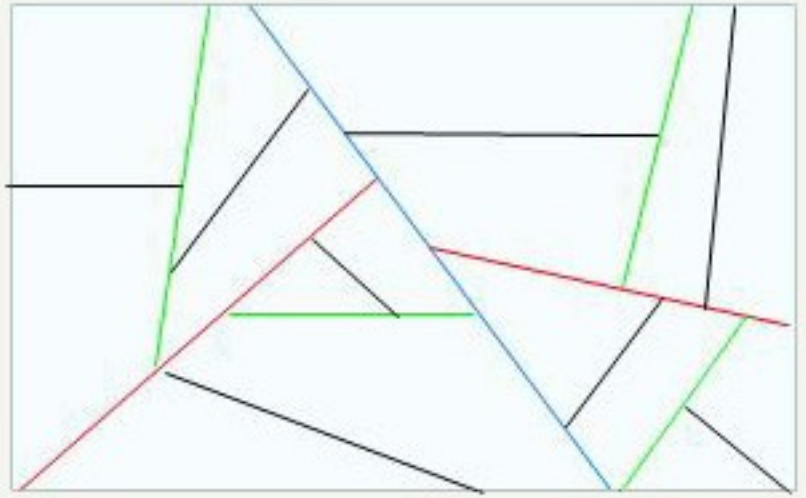
Iteration 1



Iteration 2



Iteration 3



Iteration 4

Nearest city microservice

- Given a set of entities, searches KD-tree for nearest city and inserts into entity
- Computes bearing and distance to city using Haversine- and Vincenty formula
- The enriched entity is returned to Sesam in response (JSON)

```
$ curl -s -XPOST 'http://localhost:5001/transform' -H "Content-type: application/json" -d '[{
"_id": "jane", "name": "Jane Doe", "lat": 58.995903, "lon": 10.082722}]' | jq -S .
 [
   {
     "_id": "jane",
     "lat": 58.995903,
     "lon": 10.082722,
     "name": "Jane Doe",
     "nearest_place": {
       "bearing": 170.2446453605429,
       "direction": "SSE",
       "distance": 3963.305464313277,
       "lat": 59.030966,
       "lon": 10.071019,
       "name": "Larvik",
       "postal_code": "3260"
     }
   }
 ]
```

Original entity

Added data

8 hour: Elasticsearch

- Googled how to run Elasticsearch:

```
docker pull elasticsearch

docker run --name elasticsearch -p 9200:9200 -p 9300:9300 -d elasticsearch
```

- Googled how to configure a Elasticsearch index and wrote a schema for my entities

```
{
    "mappings": {
        "ship": {
            "properties": {
                "mmsi": {"type": "integer"},
                "callsign": {"type": "string"},
                "shipname": {"type": "string"},
                "length": {"type": "integer"},
                "width": {"type": "integer"},
                "position": {"type": "string"},
                "when": {"type": "date"},
                "vendor_id": {"type": "string"},
                "url": {"type": "string"},
                "location": {"type": "geo_point"}
            }
        }
    }
} => ships.json
```

curl -XPUT http://<elasticsearch-ipaddress-here>:9200/ships @ships.json

Sesam Elasticsearch pipe

```
{
  "_id": "elasticsearch_index",
  "type": "system:elasticsearch",
  "hosts": ["172.17.0.2:9200"]
},
{
  "_id": "to_elasticsearch",
  "type": "pipe",
  "source": {
    "type": "dataset",
    "dataset": "ais_ships_with_location"
  },
  "sink": {
    "type": "elasticsearch",
    "system": "elasticsearch_index",
    "default_index": "ships",
    "default_type": "ship"
  }
}
```

A sprinkle of DTL

- I also added a bit of DTL to massage the data *slightly* on the way out

```
"transform": [
{
    "type": "dtl",
    "name": "Transform to elasticsearch document",
    "dataset": "ais_ships_with_location",
    "transforms": {
        "default": [
            ["copy", "_id"],
            ["copy", "mmsi"],
            ["add", "length", ["+", "_S.to_stern", "_S.to_bow"]],
            ["add", "width", ["+", "_S.to_port", "_S.to_starboard"]],
            ["copy", "vendor_id"],
            ["copy", "callsign"],
            ["copy", "shipname"],
            ["copy", "url"],
            ["rename", "status_text", "status"],
            ["rename", "shiptype_text", "shiptype"],
            ["merge", ["apply", "apply-last-seen", "_S.last-seen-at"]]
        ],
        "apply-last-seen": [
            ["copy", "*"],
            ["add", "location", ["dict", ["list",
                                    ["list", "lat", "_S.lat"],
                                    ["list", "lon", "_S.lon"]]
            ]],
            ["remove", "lat"],
            ["remove", "lon"]
        ]
    }
}]
```

Moment of truth – queries!

- Ships near Bergen?
  - curl -XGET 'http://172.17.0.2:9200/ships/ship/_search?q=position:bergen*&pretty=true
- Fishingboats near Leknes (in Lofoten islands, northern Norway):
  - curl -XGET 'http://172.17.0.2:9200/ships/ship/_search?q=position:leknes*%20AND%20shiptype:fishing*&pretty=true
- All ships named something with "viking":
  - curl -XGET 'http://172.17.0.2:9200/ships/ship/_search?q=shipname:viking*&pretty=true

Ships near a specific position

```
{
  "sort" : [
     {
        "_geo_distance" : {
           "location" : {
                "lat" : 59.902006,
                "lon" : 10.718077
           },
           "order" : "asc",
           "unit" : "km"
        }
     }
  ],
  "query": {
    "filtered" : {
       "query" : {
          "match_all" : {}
       },
       "filter" : {
          "geo_distance" : {
             "distance" : "5km",
             "location" : {
                "lat" : 59.902006,
                "lon" : 10.718077
             }
          }
       }
    }
  }
}
```

Success!

Using Sesam, I was able to create a working (PoC) AIS solution - in a single day! :)

(Well, *technically* over one afternoon and subsequent morning – but in sum 8 hours. honest!)

Show me the github!

- All code and full blog article

https://github.com/sesam-io/ais-integration