Data Flow

How data will be used. Examples:

ML Models

ML Models are trained on a scheduled basis. These models need training data of past few months. Example of what kind of data will be used for this training:

Example model:

- Scheduler (like a CRON job or Jenkins) will trigger model trainng .
- Model picks up configurations from `model_configs` collection.
- Trained on 6 training features : [w_force, sea_st, rpm, amb_tmp, sw_temp, sw_res] and One target feature [speed].
- Uses data of say last 6 months to train model i.e. ~180 samples.
- Total Shape of data would be 180 * (6 training+1target) = 180×7 = 1260
- So these 180 samples of 7 features would be needed for training from collection `main_db` . In `main_db` collection, Each document is a *Daily Data* for a particular ship at a particular date and contains all around~ fields for that date. Refer schema. So for last 6 months data, we need to fetch 180 documents from this collection. Each document will have 7 of those features(Out of ~200 total), so it will fetch `processed` field for each 7 of them from 180 documents.
- Once this data is fetched from collection, model is trained as per configs from `model_configs` . Logs of training will be stored in `training_logs`.Values predicted from the model will be stored in the same collection from which training data was fetched. i.e. `main_db` in the field `predictions`. If *online* predictions are also required, then this model pickle file is stored in the object storage like S3.

Plots

On a Front End, different features are shown as a time-series. Refer existing UI. Let's just take an example of current page FUEL Trends and see what kind of data will be needed there.

For Graphs

- We have total 6 Time-series subplots and each subplots hold multiple series. Noted below are all those 6 subplots with names of series. Data for these features will be fetched from `main_db`

  Let's just see fields first.

  1. speed - `processed`, `predicted.m12[0]`, `predicted.m12[1]`, `predicted.m12[2]`, `is_outlier`
  2. hfoc24- `processed`, `predicted.m12[0]`, `predicted.m12[1]`, `predicted.m12[2]`, `is_outlier`
  3. hfoc- `processed`, `predicted.m12[0]`, `predicted.m12[1]`, `predicted.m12[2]`, `is_outlier`
  4. slip- `processed`, `predicted.m12[0]`, `predicted.m12[1]`, `predicted.m12[2]`, `is_outlier`
  5. rpm - `processed`, `is_outlier`
  6. Weather
     1. sea_st - `processed`, `is_outlier`
     2. w_force - `processed`, `is_outlier`
     3. swell - `processed`, `is_outlier`
     4. current - `processed`, `is_outlier`
     5. w_dir - `processed`, `is_outlier`
     6. rel_deg - `processed`, `is_outlier`

Now, descriptions for each field:

`processed`: Cleaned values.

`predicted`: Predicted values with confidence interval. Hence the indexing.

`is_outlier`: Boolean series of denoting whether data is an outlier or not.

Total number of series required : 23 floats + 10 boolean = 33 as per above list. By default, time series is shown for 365 Days. so 365 * 33 =- 12065 unique points.

So, 365 documents will be needed to fetch and from each document those 33 fields for a day will be needed. There are few other options as well like - Comparisons which will require addional 4-5 series.

This was about graphs. Now along with graphs, tables are shown on left and right panels which show data for the day. It changes as user moves the cursor on time series for different dates.

For Tables:

Current Example: On left pane, fuels are show and on right pane other variables are shown. Now these are flexible and not yet decided and some fields might not have data for everyday. (Like Fuels). For each feature, we are showing reported and expected value (`processed` and `predicted.m12[1]`)

1. Left Table- 10 Fields - `processed`, `predicted.m12[1]`
2. Right Table -20 Fields - `processed`, `predicted.m12[1]`

Total = (20+10)*2*365 = 21k points.

This was example for Fuel Trends. Similarily we have Engine Trends and Daily Data where we have different features to be shown in a bit different format, but they all come from the same collection `main_db` .

Multiparametrics

Multiparamterics is nothing but Online Machine Learning on specific parameters. Here, trained models are prestored is static object storage lilke S3 and are used to predict values on the fly.

Because they are on the fly, data needs to be fetched from collection as per the user inputs on UI and then inference engine runs on that data. These infered values are shown on plots.

Reports

There'll be 10-15 kinds of reports. User can request data for specific duration,specific fields, and few other conditions.. This data will be presented to user in form of tables/excel/pdf files. Mostly data will be fetched from `main_db`

Discussion:

1. Which date format be used ? Few available options are MongoDB inbuilt date fields like Date() and ISODate(), or epoch format, or even a string based date ? We need flitering mostly *only* on Date and not DateTime right now, For faster querying which would be best.
2. `main_db` will be updated afterwards during calculations of predictions, outliers etc.

3. Two ways to stucture daily data in `main_db` which will heavily affect performance :

Examples (Only relevant fields shown here for brevity)

1. Nested Array of Embedded Document

```
{
   "ship_imo": 9876543,
   "date": Date("2016-05-18T16:00:00Z"),
   "ship_name": "RMTCourier",
   "data": [
   {
         "identifier":"rpm",
         "name": "RPM",
         "reported":70,
         "processed": 70,
         "is_outlier": False,
         "preprocessor_results":"Passed",
         "z_score": -2.1,
         "unit":"rpm",
         "statement":"RPM is Low",
         "predictions":{
                "m3":[71,72,73],
                "m6": [71,72,73],
                "m12": [71,72,73],
                "ly": [71,72,73],
                "dd": [71,72,73]
         }
   },
  {
         "identifier":"speed",
         "name": "Speed",
         "reported":70,
         "processed": 70,
         "is_outlier": False,
         "preprocessor_results":"Passed",
         "z_score": -2.1,
         "unit":"rpm",
         "statement":"RPM is Low",
         "predictions":{
                "m3":[71,72,73],
                "m6": [71,72,73],
                "m12": [71,72,73],
                "ly": [71,72,73],
                "dd": [71,72,73]
         }
   }
   ],
}
```

1. Nested Embedded Document

```
{
   "ship_imo": 9876543,
   "date": Date("2016-05-18T16:00:00Z"),
   "ship_name": "RMTCourier",
   "data": {
       "rpm":{
             "name": "RPM",
             "reported":70,
             "processed": 70,
             "is_outlier": False,
             "preprocessor_results":"Passed",
             "z_score": -2.1,
             "unit":"rpm",
             "statement":"RPM is Low",
             "predictions":{
                    "m3":[71,72,73],
                    "m6": [71,72,73],
                    "m12": [71,72,73],
                    "ly": [71,72,73],
                    "dd": [71,72,73]
             }
       },
      "speed":{
             "name": "Speed",
             "reported":70,
             "processed": 70,
             "is_outlier": False,
             "preprocessor_results":"Passed",
             "z_score": -2.1,
             "unit":"rpm",
             "statement":"RPM is Low",
             "predictions":{
                    "m3":[71,72,73],
                    "m6": [71,72,73],
                    "m12": [71,72,73],
                    "ly": [71,72,73],
                    "dd": [71,72,73]
             }
       }
   }
}
```

Pros and Cons of #1 and #2:

1. In #1, all individual features are collected inside `data` field. But individual fields are listed as an array of Embedded Document, not identifed by field. In this case it will be to maintain schema. As new feature is added, it will appended to the list of Embedded Documents. But a lot harder to query. Because there's no field to directly search for MongoDB.

   Say we want 'rpm' `processed` value for 365 days. So firstly, 365 documents need to be fetched for that ship. Then from each document, `data` field is

selected and then, there are 200 Embedded Documents(for 200 features) inside `data` from which document which has `identier:'rpm'` needs be extracted for getting `processed` value. This searching would take time.

MongoDB Query : (w/o dates for now)

```
db.main_db.find({"ship_imo": 9876543,"data.identifier":"rpm"},{"data.processed":1})
```

2. In #2, all individial fields have their own identifer as field name itself. So although it will harder to maintain schema if new features are added, it will be lot easier and fast to fetch the data.

   With the same example above for data of 365 days of rpm, MongoDb doesn't need to traverse through all Embedded Documents to get at rpm processed value. Directly `data.rpm.processed` would give the value.

   MongoDB Query : (w/o dates for now)

   ```
   db.main_db.findOne({"ship_imo": 9876543},{"data.rpm.processed":1});
   ```