A brief introduction to the netCDF format and THREDDS data server

Arild Burud - IT - MET Norway

WeADL 2022 Workshop

The workshop is organized under the umbrella of WeaMyL, project funded by the EEA and Norway Grants under the RO-NO-2019-0133.
Contract: No 26/2020

Topics: NetCDF - THREDDS - OPeNDAP

Acronyms - once you know them they become friends...

NetCDF: "Network Common Data Form"
TDS: "THREDDS Data Server"
THREDDS: "Thematic Real-time Environmental Distributed Data Services"
OPeNDAP: "Open-source Project for a Network Data Access Protocol"
CF Conventions: "NetCDF Climate and Forecast Metadata Conventions"
ACDD: "Attribute Convention for Dataset Discovery"
COARDS: "Cooperative Ocean/Atmosphere Research Data Service"
FIMEX: "File Interpolation, Manipulation and EXtraction library for gridded geospatial data"

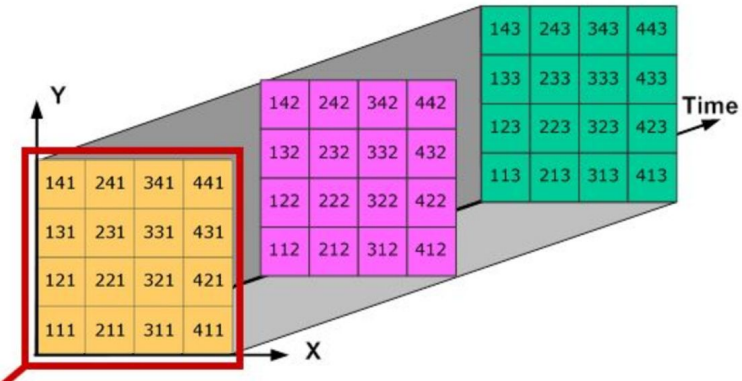
https://en.wikipedia.org/wiki/NetCDF

NetCDF (Network Common Data Form) is a set of software libraries and self-describing, machine-independent data formats that support the creation, access, and sharing of array-oriented scientific data. ... The format is an open standard. NetCDF Classic and 64-bit Offset Format are an international standard of the Open Geospatial Consortium.

The project started in 1988 and is still actively supported by UCAR. The original netCDF binary format (released in 1990, now known as "netCDF classic format") is still widely used across the world and continues to be fully supported in all netCDF releases. Version 4.0 (released in 2008) allowed the use of the HDF5 data file format. ... Version 4.7.0 (2019) added support for reading Amazon S3 objects. Version 4.8.0 (2021) with Zarr support.

Basic components of a NetCDF file

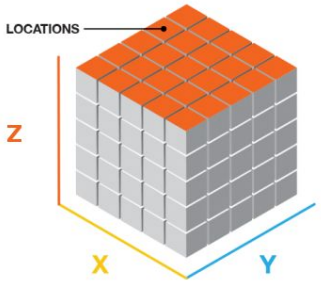
```
netcdf mynetcdf {
dimesions:
        x=4;
        y=4;
        time=UNLIMITED;
variables:
        float x(x);
        float y(y);
        int time(time);
        float temperature(time,x,y);
data:
        x = 10, 20, 30, 40;
        y = 110, 120, 130, 140;
        time = 31, 59, 90;

Temperature = 111, 211, 311, 411, 121, 221, 321,
        421, 131, 231, 331, 431, 141, 241, 341, 441;
}
```



time = 1

x = 1 to 4

y = 1 to 4



Self-describing format

```
variables:
        float lat(lat) ;
                lat:long_name = "Latitude" ;
                lat:units = "degrees_north" ;
        float lon(lon) ;
                lon:long_name = "Longitude" ;
                lon:units = "degrees_east" ;
        int time(time) ;
                time:long_name = "Time" ;
                time:units = "days since 1895-01-01" ;
                time:calendar = "gregorian" ;
        float rainfall(time, lat, lon) ;
                rainfall:long_name = "Precipitation" ;
                rainfall:units = "mm yr-1" ;
                rainfall:missing_value = -9999.f ;

// global attributes:
                :title = "Historical Climate Scenarios" ;
                :Conventions = "CF-1.0" ;

data:
 lat = 48.75, 48.25, 47.75;
 lon = -124.25, -123.75, -123.25, -122.75;
 time = 364, 730;
 rainfall =
   761, 1265, 2184, 1812, 1405, 688, 366, 269, 328, 455, 524, 877,
   1019, 714, 865, 697, 927, 926, 1452, 626, 275, 221, 196, 223;
}
```

Coordinate variable

Variable attribute

Global attribute

netCDF Conventions

Format conventions ensure cross-compatibility and makes it easier to understand/extract information. MET recommends CF Convention, ACDD, COARDS, etc... See https://www.unidata.ucar.edu/software/netcdf/conventions.html
Conventions regulate the use of attributes and naming schemes.

```
short temperature(time, depth, Y, X) ;
   temperature:units = "Celsius" ;
   temperature:standard_name = "sea_water_potential_temperature" ;
```

Data Sources

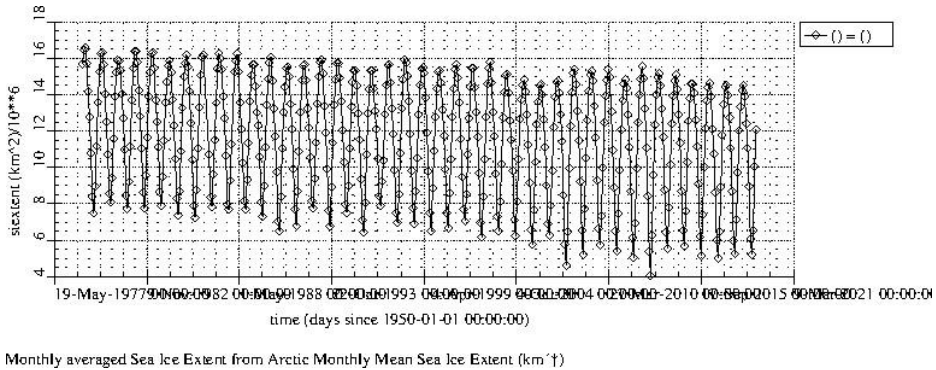
NetCDF-files come from our own production chains and from (inter)national partners, at MET Norway we store these on our lustre storage system. The files can be presented on thredds.met.no as individual files, ncml-files or as aggregated datasets. Example:

```
$ ls -l barents_opera/zdepths/
total 22893812
-rw-r--r-- 1 xxx yyy 3471331596 Jun 12 11:43 Barents-2.5km_ZDEPTHS_his.an.2020061100.nc
-rw-rw-r-- 1 xxx yyy 3271502854 Jun 15 07:50 Barents-2.5km_ZDEPTHS_his.an.2020061200.nc
-rw-rw-r-- 1 xxx yyy 3274018343 Jun 15 07:53 Barents-2.5km_ZDEPTHS_his.an.2020061300.nc
-rw-rw-r-- 1 xxx yyy 3475706944 Jun 15 21:21 Barents-2.5km_ZDEPTHS_his.an.2020061400.nc
-rw-rw-r-- 1 xxx yyy 9950665619 Jun 15 21:21 Barents-2.5km_ZDEPTHS_his.fc.nc
```

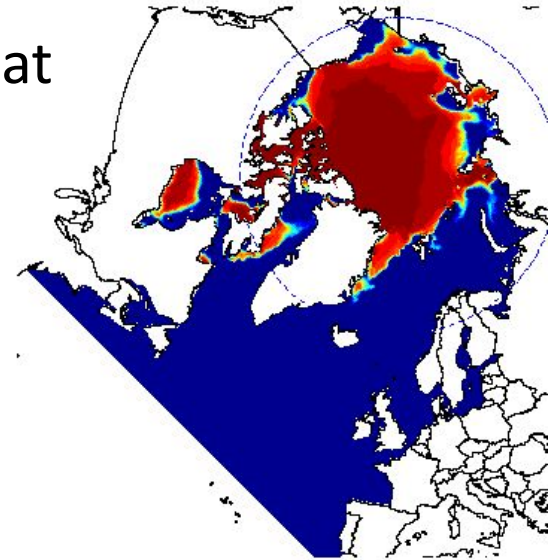
thredds.met.no

https://thredds.met.no is using the TDS software from Unidata/UCAR.

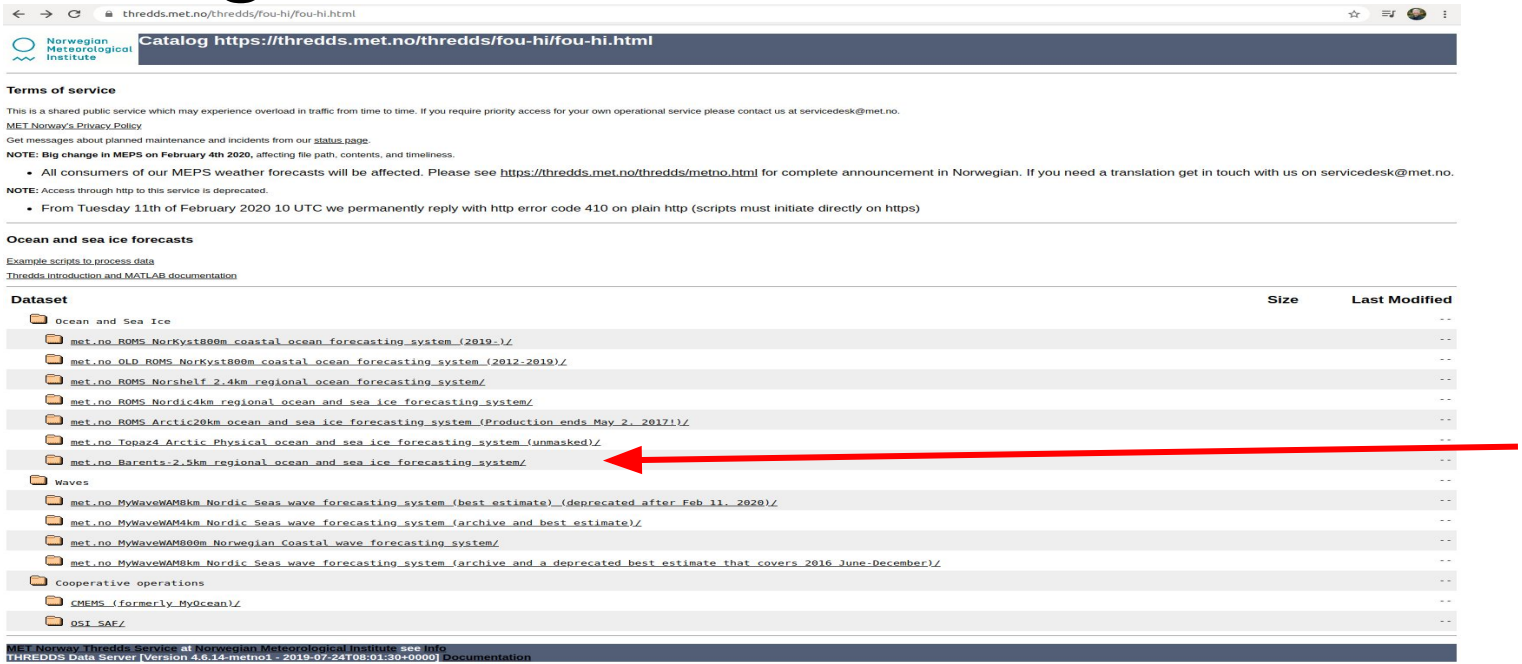
- We use it to publish data in NetCDF format
- Typically gridded data or timeseries



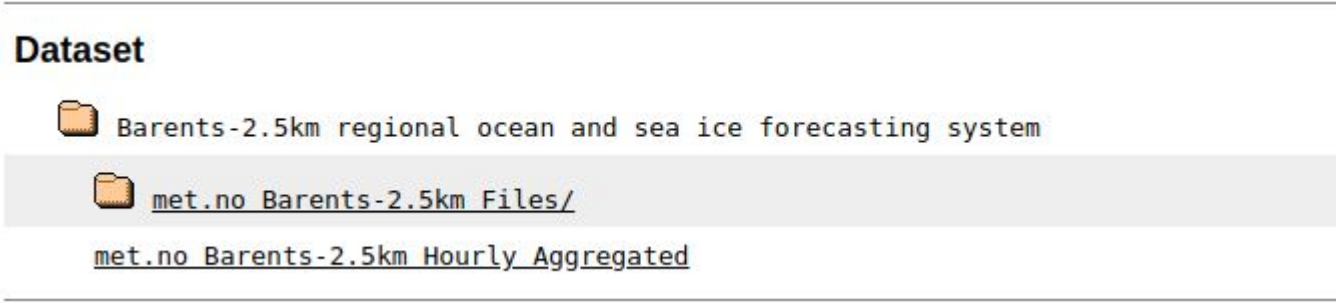
Monthly averaged Sea Ice Extent from Arctic Monthly Mean Sea Ice Extent (km^2)



Looking at thredds.met.no



Web view



Many datasets can be accessed as either individual files or an aggregation of all files along the time axis. Imagine stitching together 24 hour spans and presenting it as a complete file (although not downloadable as a file) that cover years of results.

Web view - individual files

Dataset

Barents-2.5km regional ocean and sea ice forecasting system

met.no Barents-2.5km Files/

met.no Barents-2.5km Hourly Aggregated

Dataset	Size	Last Modified
met.no Barents-2.5km Files		--
Barents-2.5km_ZDEPTHS_his.fc.nc	9.950 Gbytes	2020-06-15T21:21:36Z
Barents-2.5km_ZDEPTHS_his.an.2020061400.nc	3.475 Gbytes	2020-06-15T21:21:11Z
Barents-2.5km_ZDEPTHS_his.an.2020061300.nc	3.274 Gbytes	2020-06-15T07:53:40Z
Barents-2.5km_ZDEPTHS_his.an.2020061200.nc	3.271 Gbytes	2020-06-15T07:50:07Z
Barents-2.5km_ZDEPTHS_his.an.2020061100.nc	3.471 Gbytes	2020-06-12T11:43:02Z

Individual files

MET Norway Thredds Service

THREDDS Data Server

Catalog https://thredds.met.no/thredds/catalog/barents25km_files/catalog.html

Dataset: met.no Barents-2.5km Files/Barents-2.5km_ZDEPTHS_his.fc.nc

- *Data size:* 9.950 Gbytes
- *ID:* barents25km_files/Barents-2.5km_ZDEPTHS_his.fc.nc

Access:

1. **OPENDAP:** /thredds/dodsC/barents25km_files/Barents-2.5km_ZDEPTHS_his.fc.nc
2. **HTTPServer:** /thredds/fileServer/barents25km_files/Barents-2.5km_ZDEPTHS_his.fc.nc
3. **WMS:** /thredds/wms/barents25km_files/Barents-2.5km_ZDEPTHS_his.fc.nc
4. **WCS:** /thredds/wcs/barents25km_files/Barents-2.5km_ZDEPTHS_his.fc.nc

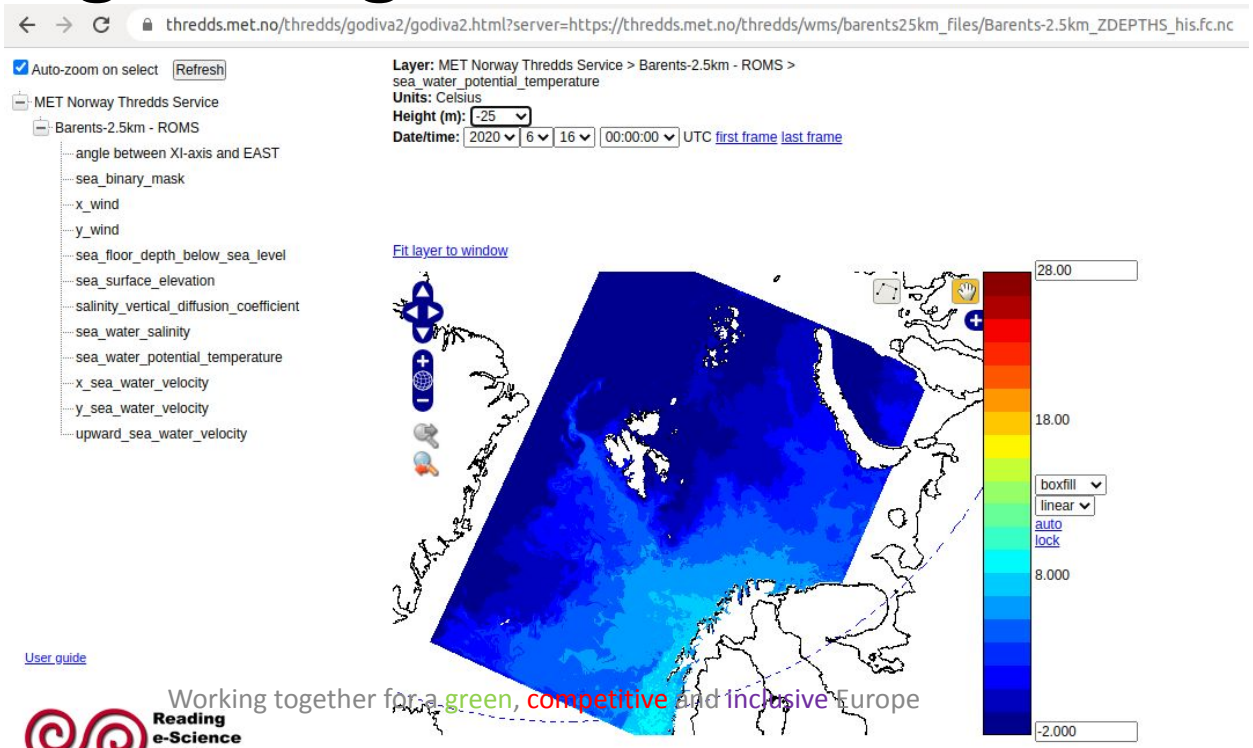
Dates:

- 2020-06-15T21:21:36Z **(modified)**

Viewers:

- Godiva2 (browser-based)
- NetCDF-Java ToolsUI (webstart)

Visualizing through Godiva2



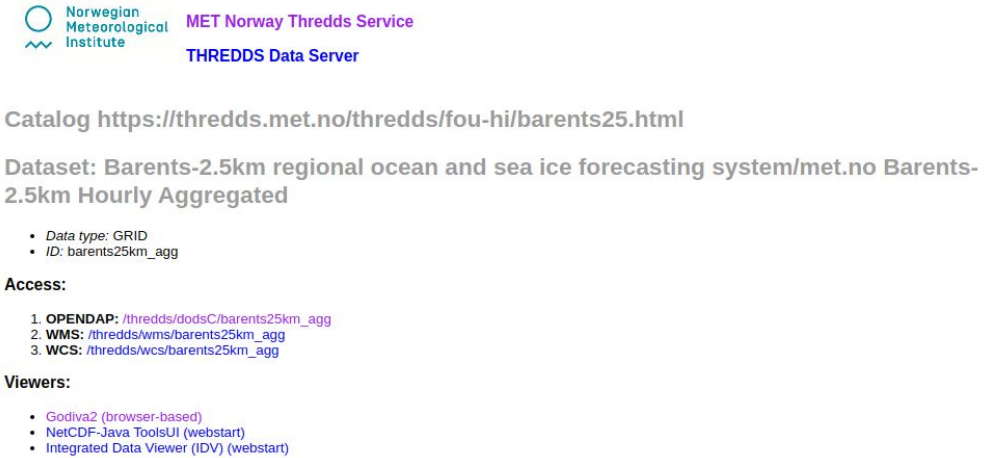
Web view - aggregation

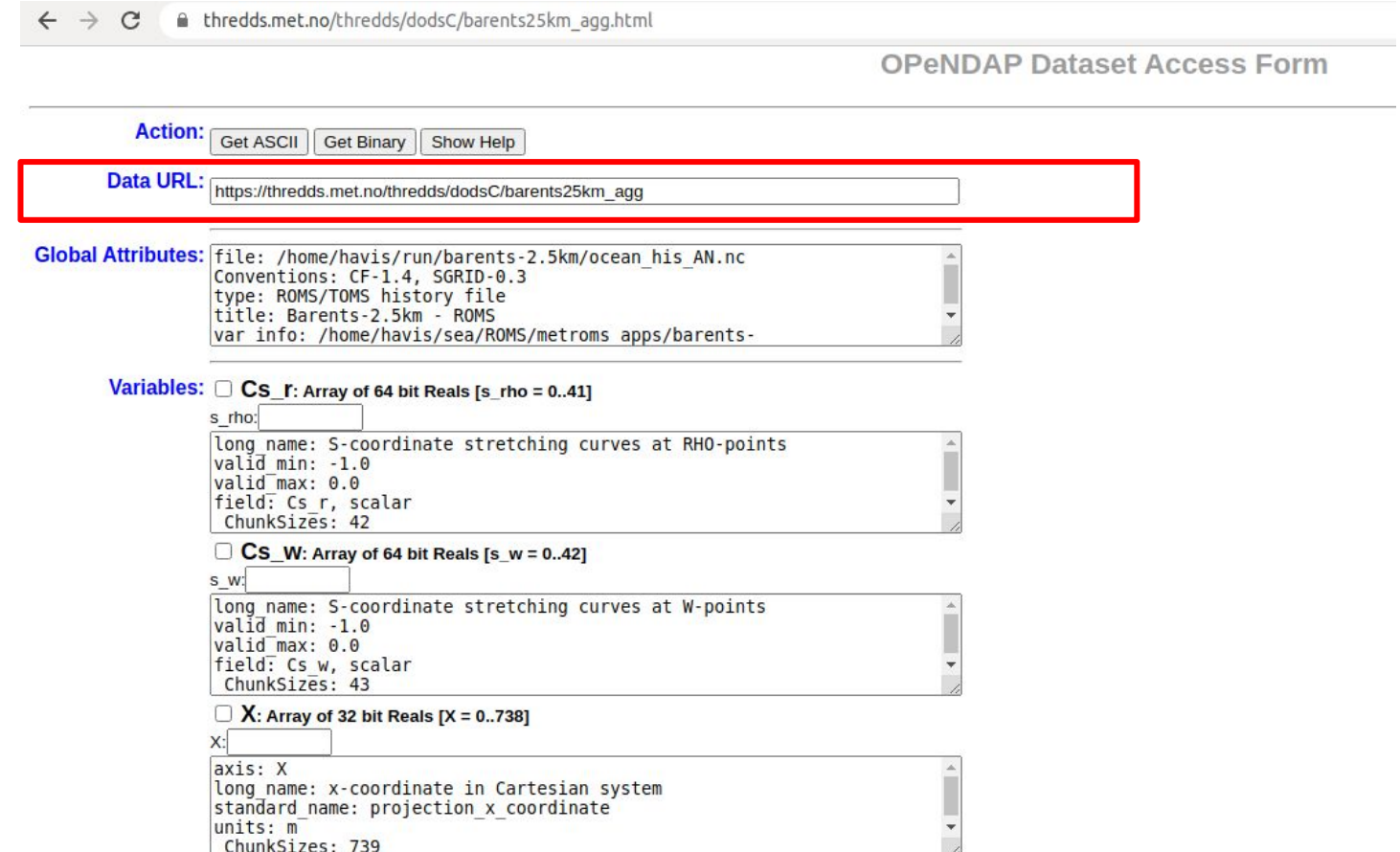
Dataset

Barents-2.5km regional ocean and sea ice forecasting system

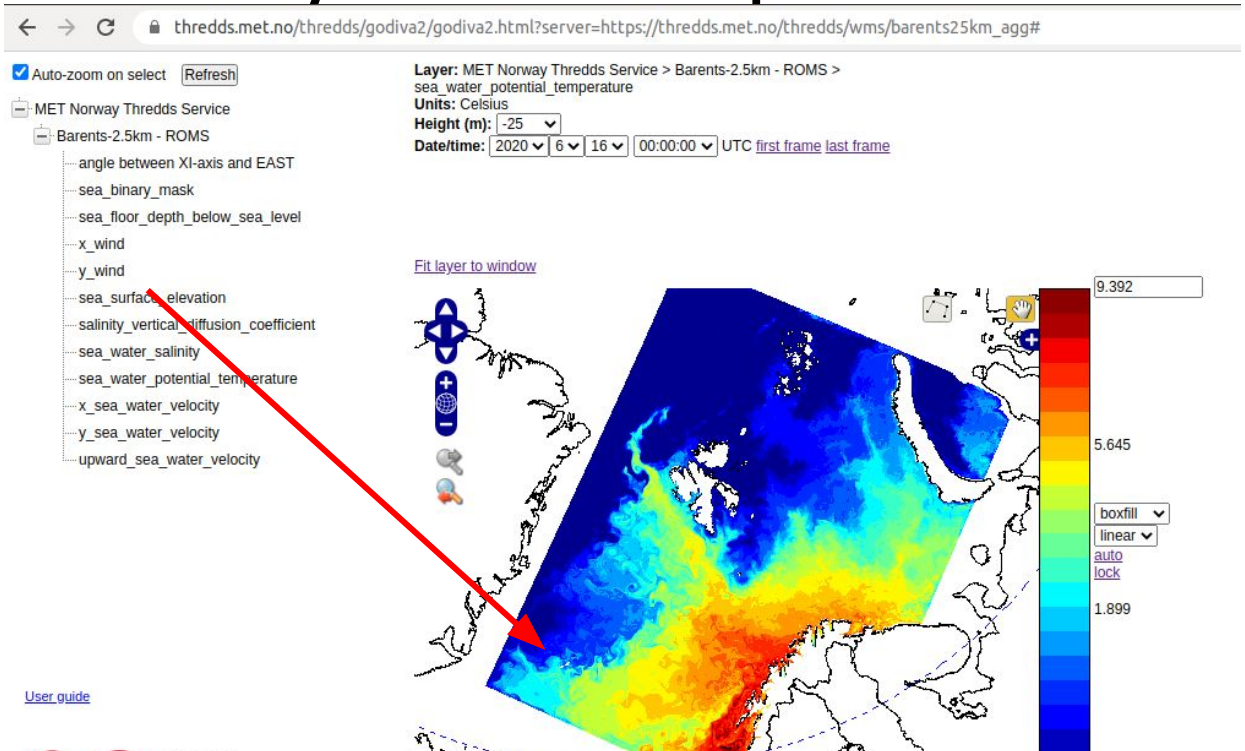
met.no Barents-2.5km Files/

met.no Barents-2.5km Hourly Aggregated



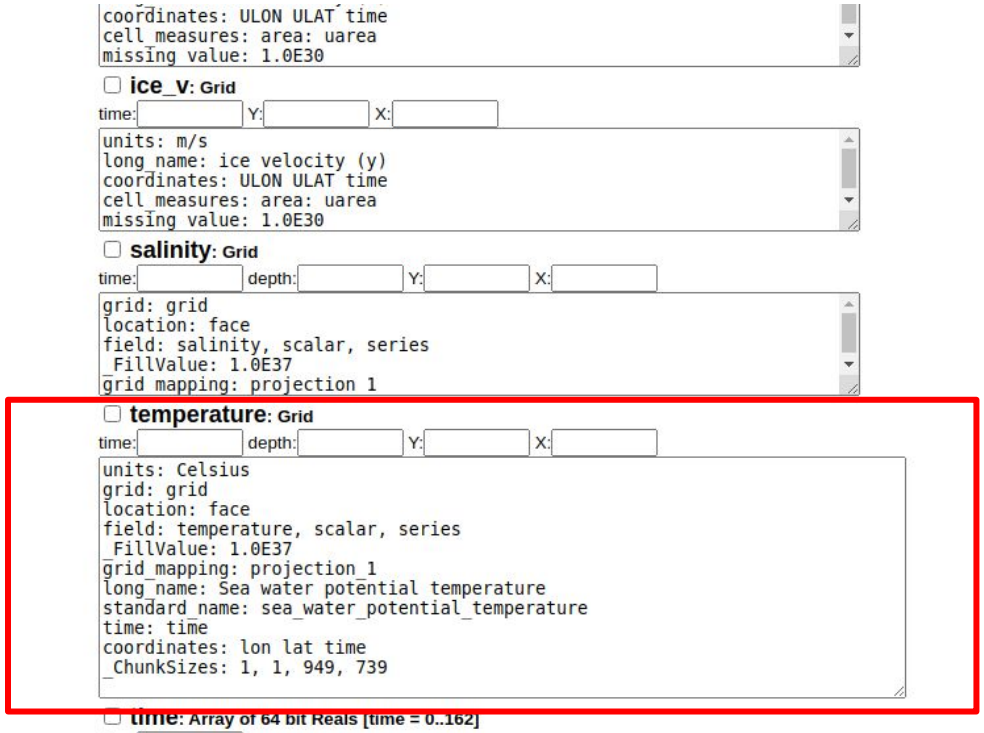

thredds.met.no/thredds/dodsC/barents25km_agg.html
OPeNDAP Dataset Access Form
Action:
Get ASCII
Get Binary
Show Help
Data URL:
https://thredds.met.no/thredds/dodsC/barents25km_agg
Global Attributes:
file: /home/havis/run/barents-2.5km/ocean_his_AN.nc
Conventions: CF-1.4, SGRID-0.3
type: ROMS/TOMS history file
title: Barents-2.5km - ROMS
var_info: /home/havis/sea/ROMS/metroms_apps/barents-
Variables:
Cs_r: Array of 64 bit Reals [s_rho = 0..41]
s_rho:
long_name: S-coordinate stretching curves at RHO-points
valid_min: -1.0
valid_max: 0.0
field: Cs_r, scalar
ChunkSizes: 42
Cs_W: Array of 64 bit Reals [s_w = 0..42]
s_w:
long_name: S-coordinate stretching curves at W-points
valid_min: -1.0
valid_max: 0.0
field: Cs_w, scalar
ChunkSizes: 43
X: Array of 32 bit Reals [X = 0..738]
X:
axis: X
long_name: x-coordinate in Cartesian system
standard_name: projection_x_coordinate
units: m
ChunkSizes: 739

Find Jan Mayen seatemp at 25m



Through OPeNDAP webpage ?

Needs index 0-N



Use fimex with natural coordinates

Jan Mayen is located near 70N, 8W

```
$ fimex
--input.file=https://thredds.met.no/thredds/dodsC/barents25km_agg
--input.type=netcdf --output.type=nc4
--extract.selectVariables=temperature
--extract.reduceToBoundingBox.south 69.
--extract.reduceToBoundingBox.north 72.
--extract.reduceToBoundingBox.west -10.
--extract.reduceToBoundingBox.east -7.
--extract.reduceVerticalAxis.start=20.
--extract.reduceVerticalAxis.end=30.
--extract.reduceVerticalAxis.unit=m
--extract.reduceTime.start=2020-06-16T00:00:00
--extract.reduceTime.end=2020-06-16T23:00:00
--output.file=mysubset.nc4

$ ls -l mysubset.nc4
-rw-rw-r-- 1 myid group 900475 Jun 16 08:04 mysubset.nc4
```

Notice: original 24 hour file was >3GB

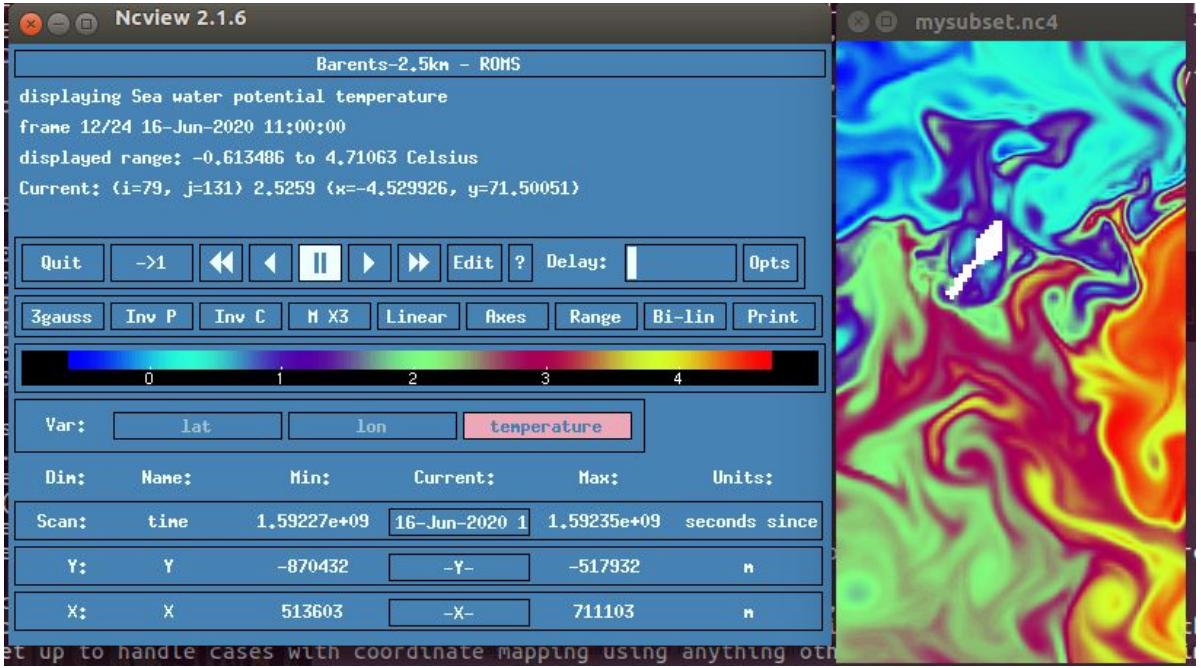
What does it look like?

```
$ ncdump -h mysubset.nc4
netcdf mysubset {
dimensions:
        X = 80 ;
        Y = 142 ;
        depth = 1 ;
        time = 24 ;
variables:
        float X(X) ;
                X:axis = "X" ;
                X:long_name = "x-coordinate in Cartesian system" ;
                X:standard_name = "projection_x_coordinate" ;
                X:units = "m" ;
        float Y(Y) ;
                Y:axis = "Y" ;
                Y:long_name = "y-coordinate in Cartesian system" ;
                Y:standard_name = "projection_y_coordinate" ;
                Y:units = "m" ;
        double depth(depth) ;
                depth:units = "m" ;
                depth:positive = "down" ;
                depth:axis = "Z" ;
                depth:standard_name = "depth" ;
        int projection_1 ;
                projection_1:grid_mapping_name =
"lambert_conformal_conic" ;
                projection_1:proj4 = "+proj=lcc +lat_0=77.5 +lon_0=-25
+lat_1=77.5 +lat_2=77.5 +no_defs +R=6.371e+06" ;
        double time(time) ;
                time:long_name = "time since initialization" ;
                time:units = "seconds since 1970-01-01 00:00:00" ;
                time:calendar = "gregorian" ;
                time:field = "time, scalar, series" ;
                time:axis = "T" ;
                time:standard_name = "time" ;
```

```
        double lat(Y, X) ;
                lat:long_name = "latitude of RHO-points" ;
                lat:units = "degree_north" ;
                lat:standard_name = "latitude" ;
                lat:grid_mapping = "projection_1" ;
                lat:field = "lat, scalar" ;
        double lon(Y, X) ;
                lon:long_name = "longitude of RHO-points" ;
                lon:units = "degree_east" ;
                lon:standard_name = "longitude" ;
                lon:grid_mapping = "projection_1" ;
                lon:field = "lon, scalar" ;
        float temperature(time, depth, Y, X) ;
                temperature:units = "Celsius" ;
                temperature:grid = "grid" ;
                temperature:location = "face" ;
                temperature:field = "temperature, scalar, series" ;
                temperature:_FillValue = 1.e+37f ;
                temperature:grid_mapping = "projection_1" ;
                temperature:long_name = "Sea water potential temperature" ;
                temperature:standard_name =
"sea_water_potential_temperature" ;
                temperature:time = "time" ;
                temperature:coordinates = "lon lat time" ;
```

What does it look like?

```
$ ncview mysubset.nc4
```



Benefits?

- OPeNDAP URL for aggregation does not change as new files enter from daily production - no need to decode filenames
- Fimex allows natural coordinates (lat/lon, depth, ISO-time) instead of indexes
- Extract just what you need (900KB from a 20GB dataset)
- Thredds makes data available everywhere, portability of data use (lustre -> thredds -> fimex -> HPC)
- Fimex can also regrid and interpolate

See https://wiki.met.no/fimex/documentation - in particular the workshop presentations

Other solutions?

Fimex is not the only tool that can abstract OPeNDAP access, see NCKS, NCO, Perl, Python, Ruby, R, MATLAB, IDL, etc. The list of tools is growing.
https://www.unidata.ucar.edu/software/netcdf/software.html

You may find anything from full GUI applications (IDV), command-line tools, to libraries that you can embed in your favourite programming language. Most certainly a toolbox is ready for you in your favourite working environment.

So what's behind thredds.met.no?

MET Norway: High redundancy setup with two data rooms

Traffic through thredds.met.no is handled by one load balancer in each data room, using a docker image with nginx implemented in a k8s environment. Each load balancer can access 11 TDS machines in each data room - for a total of 22 identically configured TDS machines. Each TDS machine runs a TDS implemented in a docker image running in an ostack environment. The configuration is controlled through gitlab with CI/CD integration. This gives us a high redundancy, easy failover and high throughput.

thredds.met.no key figures

Total output April 2022 : 466TB

Daily output: 10-20 TB

Daily users (eg. IP-adresses): 1000+

Daily requests: 5-10 million

Approx 70% of output is OpeNDAP

Approx 30% of output is raw file download

Exposing selected parts of approx 10 PB of data from our lustre file storage system

TDS for WeaMyL

MET supplied a docker image with a minimal configuration and assisted in spinning up this for WeaMyL. It was a relatively simple task to expose a directory of netCDF files through the installation:

```
/usr/bin/docker run ...
-v /home/weamyl/WEAMYL_project/out/RADAR:/weamyl_data:rslave
metno/thredds:production
```

Summary

- Create your data files as NetCDF, add CF Conventions to describe the content and make it easy to read.
- Expose the files through TDS, you can make aggregations and collections. You get a built-in viewer, OPeNDAP access and WMS out-of-the-box
- Use OPeNDAP-enabled tools to extract what you need and process the data
- Users are happy and the producers don't need to push data around

Questions?