Earthkit Introduction

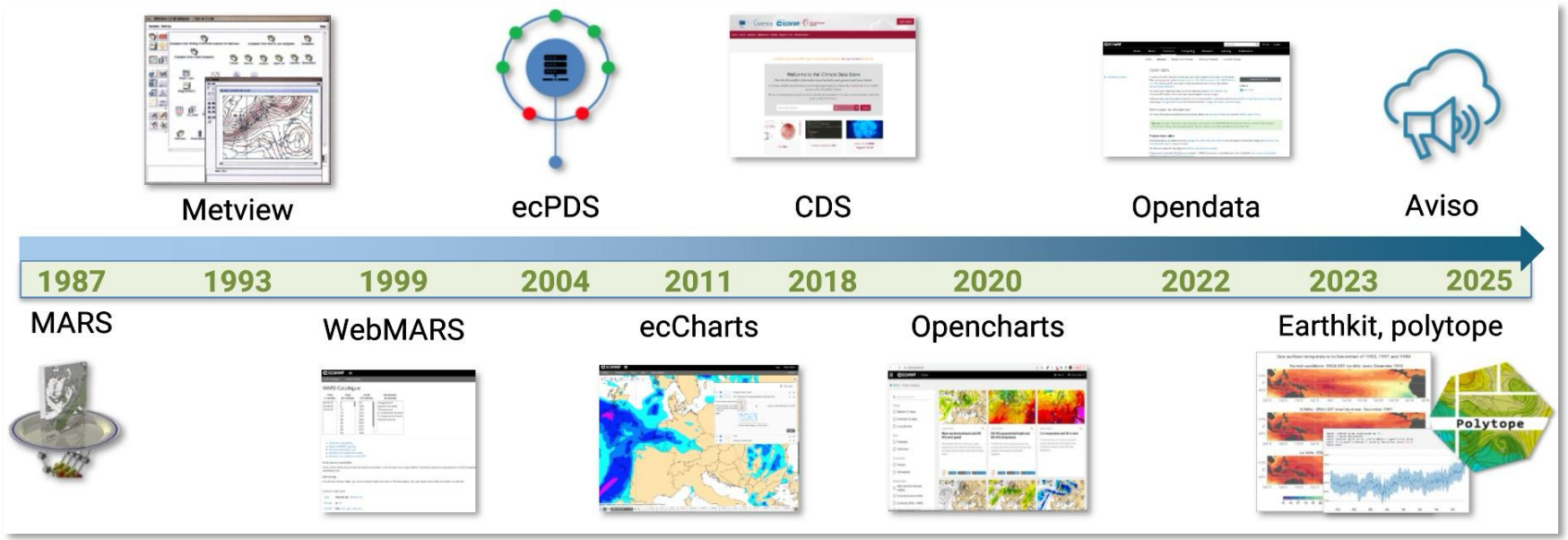
Online training week 2025

Iain Russell, Sándor Kertész, James Varndell

ECMWF

Context of ECMWF software development

- To support its NWP models, ECMWF has been developing data-centric software for decades for production and users ('*ECMWF Software Engine*')



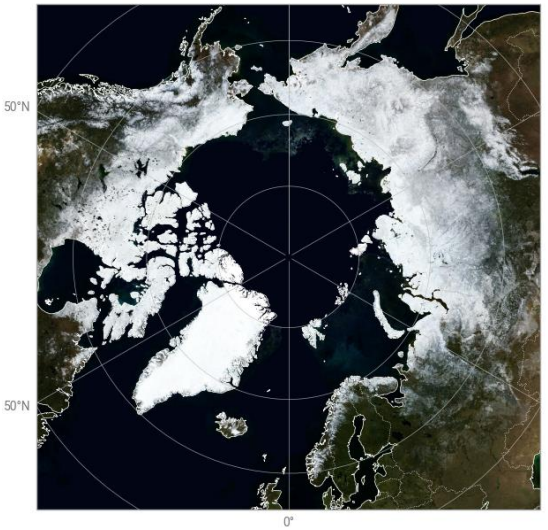
- Time for renewal
  - ECMWF's "Software Strategy and Roadmap for 2023–2027" planned a major refactoring of our software stack to improve:
    - Open development
    - Reusability and componentisation
    - Use of third-party software
    - Technical considerations such as scalability on modern hardware

Introducing Earthkit

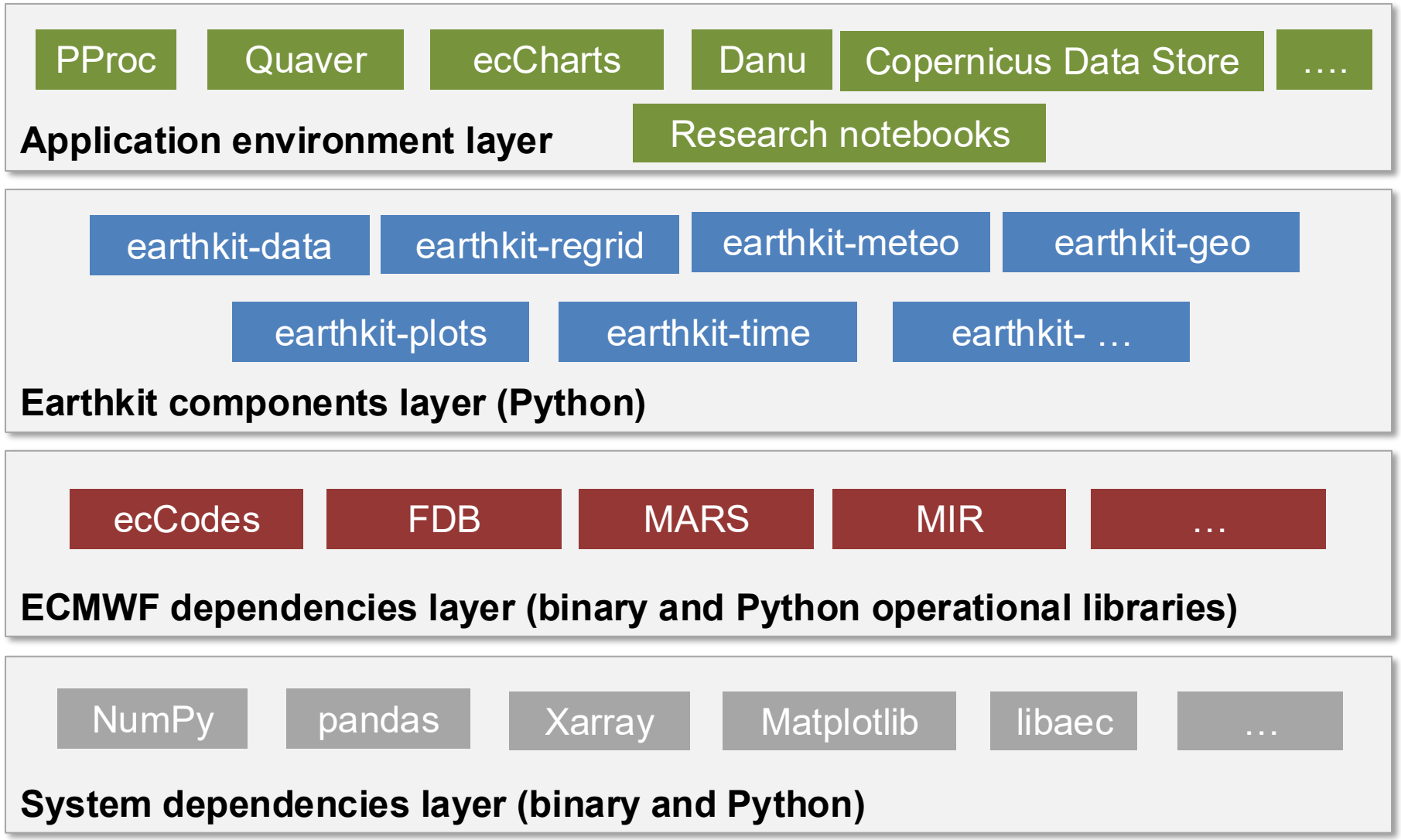
- **Earthkit**
  - New set of high-level scalable, interoperable, focused Python components
  - Suitable for use by our **operational services** and directly by **researchers / analysts**
  - Designed with Machine Learning / GPU / In-memory computations as *first-class citizens*
  - Designed with diskless data access in mind
  - Reduce boilerplate code
  - *Components are in different stages of maturity, but some are already well-tested and in operational use at ECMWF*

ESEE Foundation | Maturity Incubating | License Apache 2.0 | Release v0.17.0

- **Open Development**
  - Highly collaborative both inside and outside ECMWF
  - All code is on GitHub, fully embracing Open Development



Earthkit's place in the ECMWF software stack



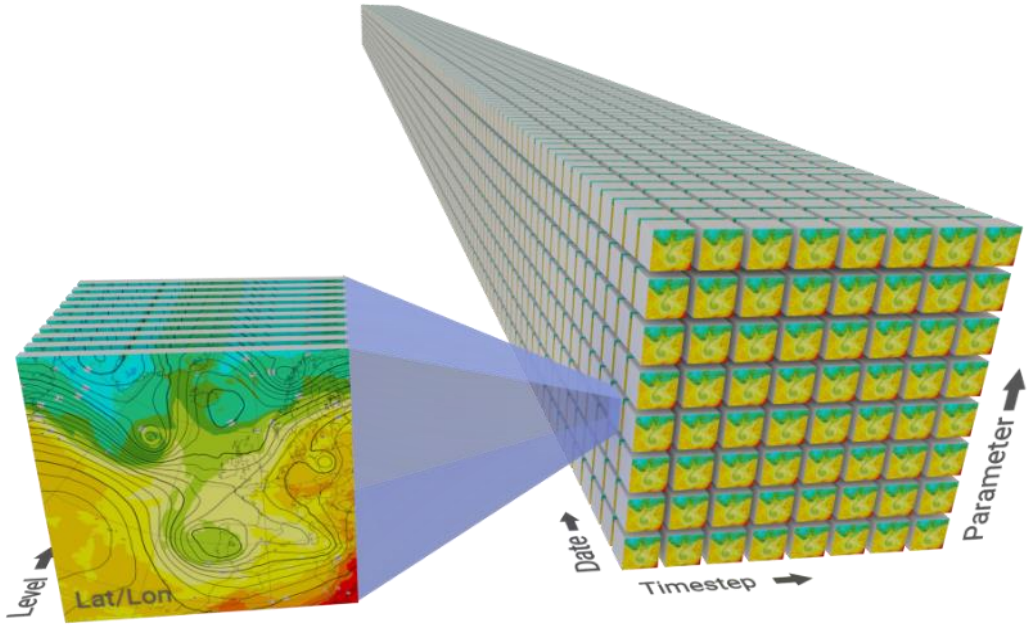
(our aim)

Earthkit and its Python components

Core components

earthkit data

- Format-agnostic handling of geospatial data from local or remote files, and services; **access**, **inspection**, **filtering**, **safe modification**
- Field data (horizontal slice of the atmosphere)
  - A *field* is abstracted to an array of values plus a dict-like set of metadata
  - From GRIB, NetCDF, GeoTiff, shapefiles, NumPy
  - can be accessed as either a 'list' of fields or converted to NumPy, pandas or Xarray (using earthkit-data's **new GRIB->Xarray engine**)
- Non-field data
  - From BUFR, ODB, CSV, CoverageJSON
  - will be converted to pandas or Xarray



Example: streaming data from an S3 bucket via earthkit-data

```python
import earthkit.data

req = {"endpoint": "object-store.os-api.cci1.ecmwf.int",
       "bucket": "earthkit-test-data-public",
       "objects": "step6.grib",
   }

ds = earthkit.data.from_source("s3", req, stream=True, read_all=True, anon=True)

ds.ls()
```

	centre	shortName	typeOfLevel	level	dataDate	dataTime	stepRange
0	ecmf	2t	surface	0	20250106	1200	6
1	ecmf	10fg6	surface	0	20250106	1200	0-6

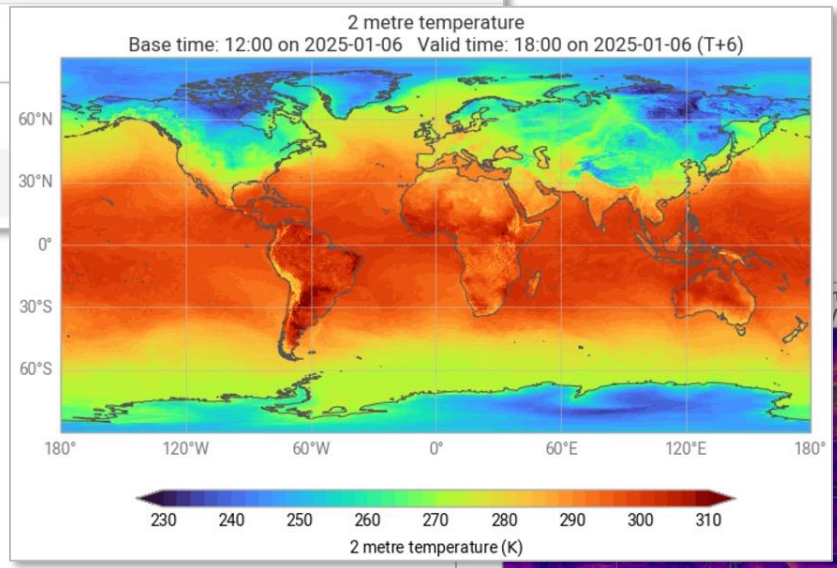
`stream`=`False`: write to disk cache

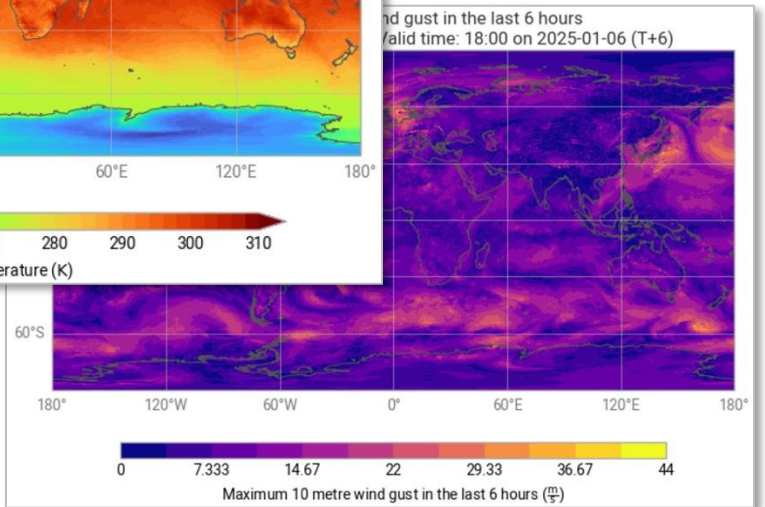
`stream`=`True`: read into memory

`read_all`=`False`: return iterator to load one field at a time

`read_all`=`True`: load all fields into memory

```python
import earthkit.plots

for f in ds:
    earthkit.plots.quickmap.block(f)
```





Getting data from files, URLs and services via earthkit-data

```python
import earthkit.data
```

```python
ds = earthkit.data.from_source("file", "test.nc")
```

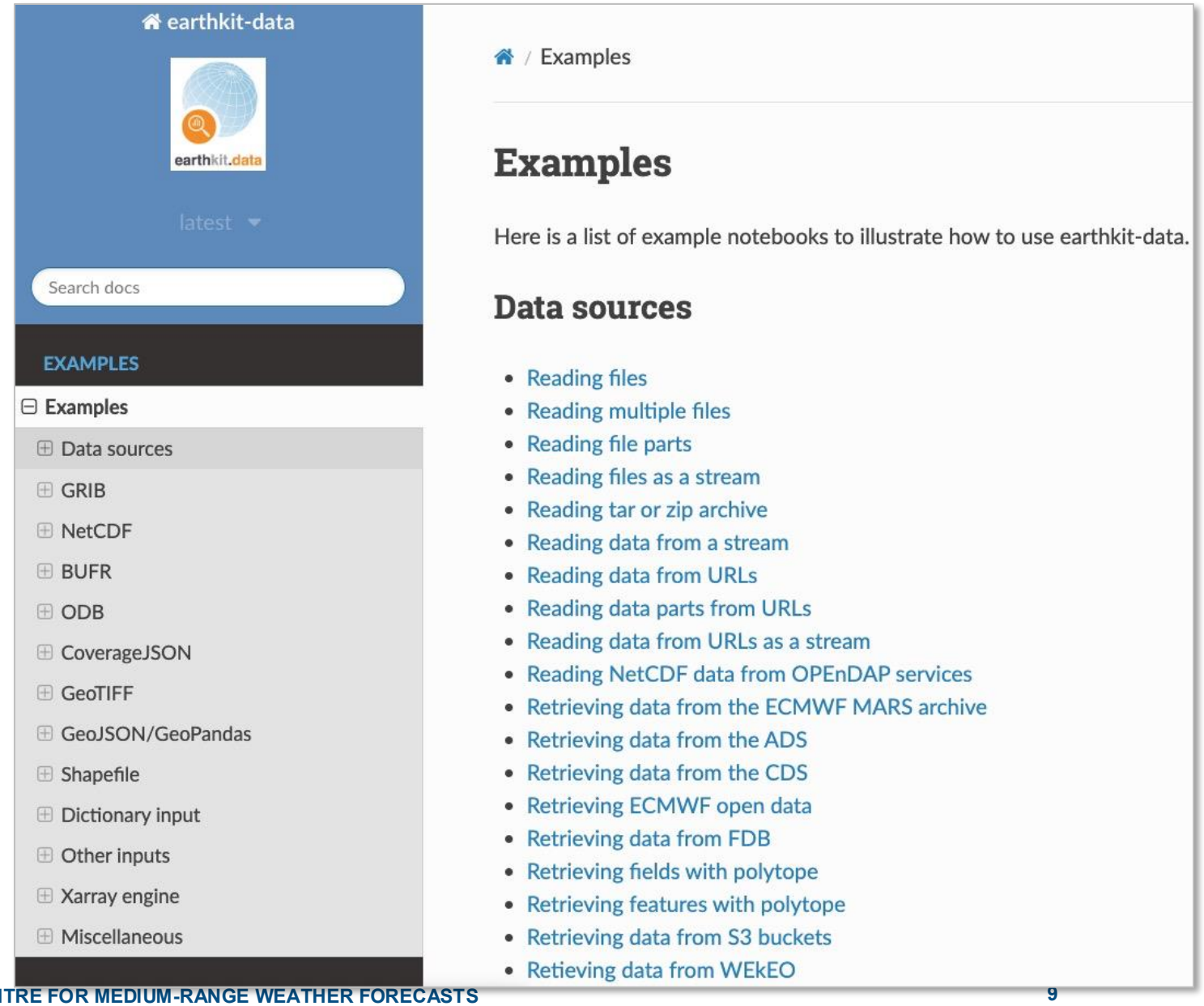
```python
ds = earthkit.data.from_source("url",
    "https://get.ecmwf.int/repository/test-data/earthkit-data/test-data/t_pl.grib")
```

```python
ds = earthkit.data.from_source("cds",
        "reanalysis-era5-single-levels",
        variable=["2t", "msl"],
        product_type="reanalysis",
        area=[50, -10, 40, 10], #N,W,S,E
        grid=[2, 2],
        date="2012-05-10",
        time="12:00",
        format="netcdf")
```

```python
ds = earthkit.data.from_source("ecmwf-open-data",
        param=["t", "gh"],
        levelist="500",
        step=[0,6,12])
```

Get more from earthkit-data

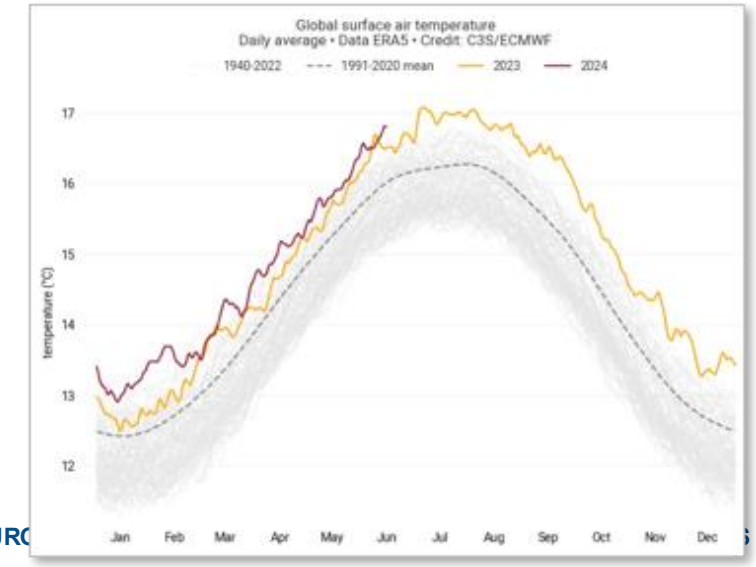
- Many data formats and ways to read them!

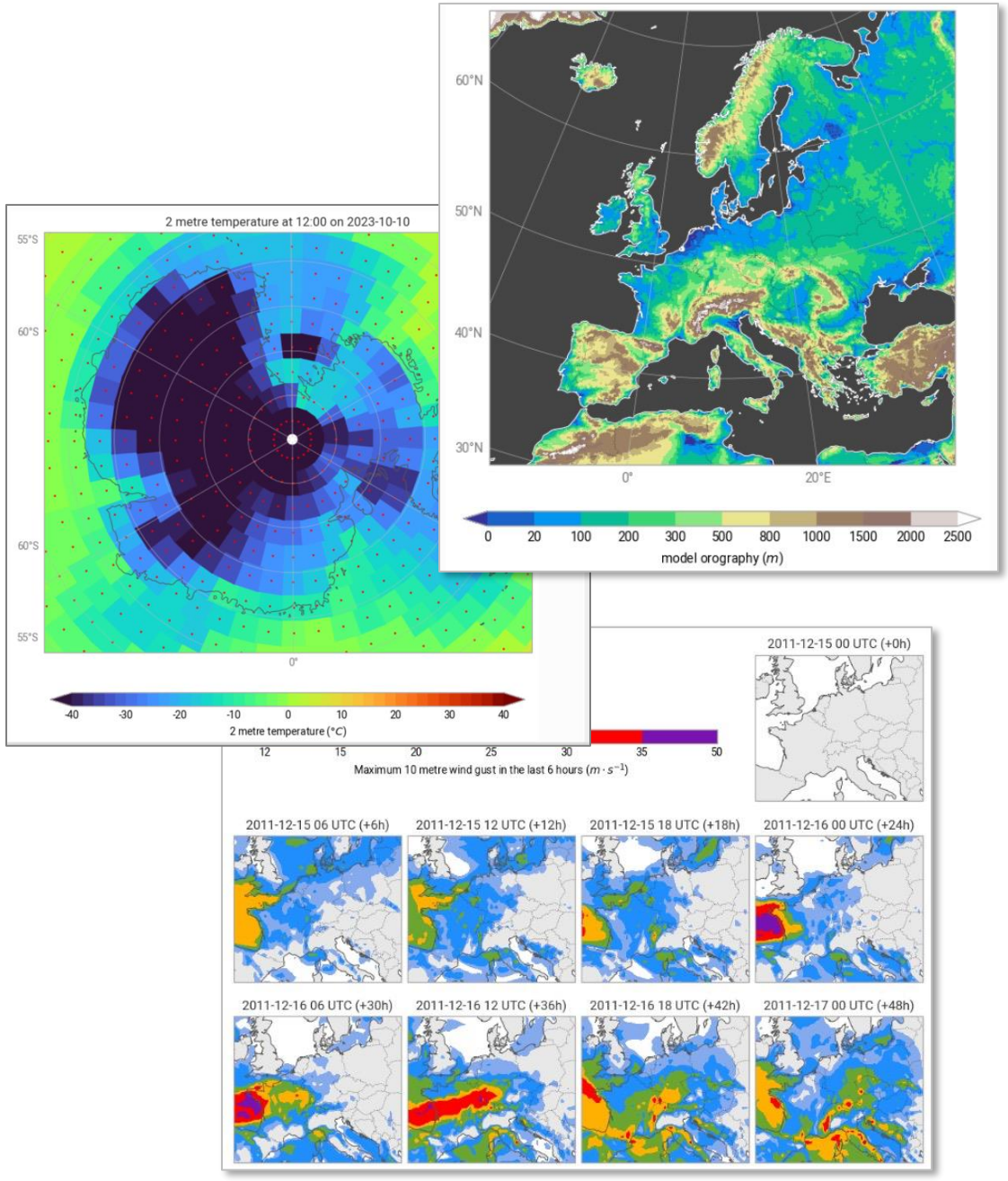


Core components

earthkit plots

- Publication-quality weather and climate graphs and maps from **NumPy**, **Xarray** and **earthkit-data** objects
- Based on Matplotlib, cartopy (and Plotly)
- Domain-specific knowledge; built-in, extendable styles; reduces boilerplate



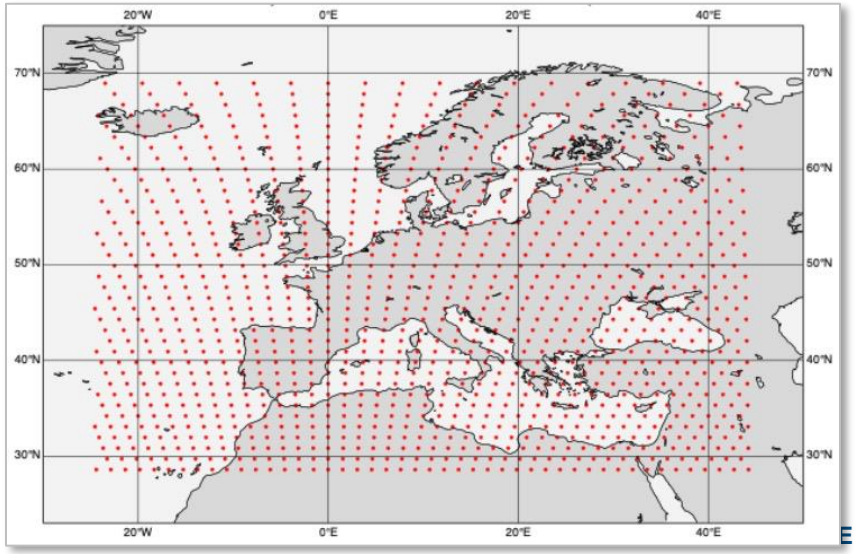


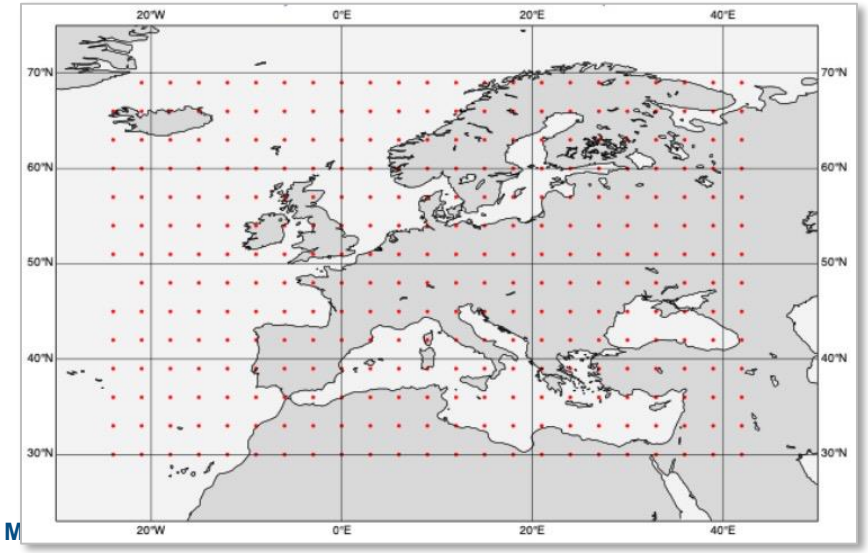
Geospatial

earthkit regrid

- Pure Python implementation of regridding of data, e.g. reduced Gaussian grid to regular lat/lon grid
- Inputs can be NumPy arrays or earthkit-data objects
- New functionality coming when we link it with our MIR library

```python
import os
from earthkit.regrid import interpolate
from earthkit.data import from_source

# Get octahedral reduced Gaussian GRIB data containing two fields.
ds = from_source(
    "url",
    "https://get.ecmwf.int/repository/test-data/earthkit-regrid/examples/O32_multi.grib2")

# the target grid is a global 5x5 degree regular latitude-longitude grid
out_grid = {"grid": [5,5]}

# perform interpolation for each field and add results
# to a new fieldlist stored in memory
r = interpolate(ds, out_grid=out_grid, method="linear")
```



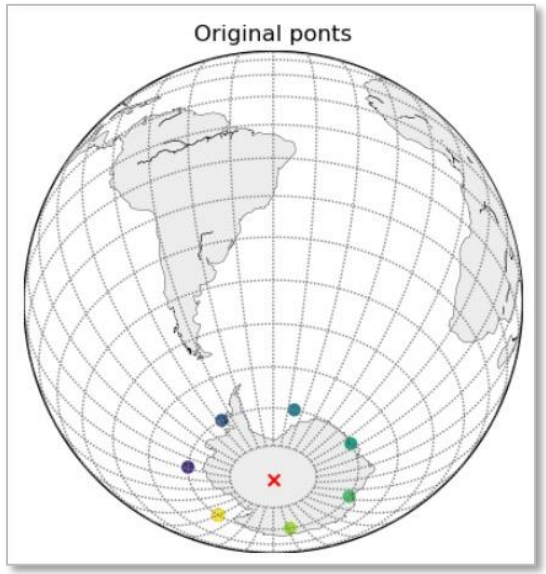


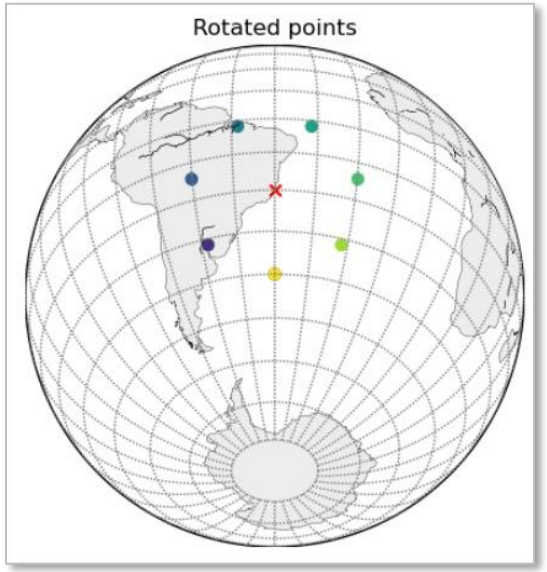
Geospatial

earthkit geo

- Geographic manipulations

```python
import numpy as np
from earthkit.geo.rotate import rotate

# new position of the south pole, this defines the rotation
south_pole = [-20, -40]

# list of points on the lat=-70 latitude
lat = np.array([-70]*8)
lon = np.linspace(-180, 180, 8)

# perform rotation
lat_r, lon_r = rotate(lat, lon, south_pole[0], south_pole[1])

# plot the points, red cross marks the south pole
plot_globe([lat, lon], [-90, 0, "r", "x"], title="Original ponts")
plot_globe([lat_r, lon_r], [-20, -40, "r", "x"], title="Rotated points")
```





Computations

earthkit meteo

- Meteorological computations
- Thermodynamics, wind, CRPS, EFI
- Vertical
- Currently array in/out; earthkit-data objects and xarray will be directly supported in a future release

```python
from earthkit.meteo import thermo
import numpy as np

t = np.array([264.12, 261.45])  # Kelvins
p = np.array([850, 850]) * 100.0  # Pascals

theta = thermo.potential_temperature(t, p)
```

Functions

`celsius_to_kelvin` (t)	Converts temperature values from Celsius to Kelvin.
`dewpoint_from_relative_humidity` (t, r)	Compute the dewpoint temperature from relative humidity.
`dewpoint_from_specific_humidity` (q, p)	Compute the dewpoint temperature from specific humidity.
`ept_from_dewpoint` (t, td, p[, method])	Computes the equivalent potential temperature from dewpoint.
`ept_from_specific_humidity` (t, q, p[, method])	Computes the equivalent potential temperature from specific humidity.
`kelvin_to_celsius` (t)	Converts temperature values from Kelvin to Celsius.
`lcl` (t, td, p[, method])	Computes the temperature and pressure of the Lifting Condenstaion Level (LCL) from dewpoint.
`lcl_temperature` (t, td[, method])	Computes the Lifting Condenstaion Level (LCL) temperature from

Computations

earthkit transforms

- Aggregations in time and space

The median over the time dimension

```
[6]: era5_t_median = ek_aggregate.temporal.reduce(era5_data, how="median")
     era5_t_median
```

[6]: xarray.Dataset

▸ Dimensions: (**number**: 1, **step**: 1, **surface**: 1, **latitude**: 201, **longitude**: 281)

(number)	int64	0	
(step)	timedelta64[ns]	0...	
(surface)	float64	0.0	
(latitude)	float64	8...	
(longitude)	float64	-...	
(number, step, surface, latitude, longitude)	float32	2...	

transforms.aggregate.climatology.daily_max(*dataarray, *args, **kwargs)

Calculate the daily climatological max.

Parameters:

- **dataarray** (`xr.DataArray`) – The DataArray over which to calculate the climatological max. Must contain a *time* dimension.
- **bin_widths** (`int` or `list (optional)`) – If *bin_widths* is an *int*, it defines the width of each group bin on the frequency provided by *frequency*. If *bin_widths* is a sequence it defines the edges of each bin, allowing for non-uniform bin widths.
- **time_dim** (`str (optional)`) – Name of the time dimension in the data object, default behaviour is to detect the time dimension from the input object
- ****reduce_kwargs** – Any other kwargs that are accepted by *earthkit.transforms.aggregate.reduce* (except how)

Return type: `xr.DataArray`

Temporal

earthkit time

- Manipulations of dates and time for weather forecasting and climatology

Sequence examples

Example	Description
`DailySequence()`	Sequence recurring every day
`DailySequence(excludes=[31])`	Sequence recurring every day, except the 31st
`WeeklySequence([MONDAY, THURSDAY])`	Sequence recurring every Monday and Thursday
`MonthlySequence([1, 15])`	Sequence recurring every 1st and 15th of the month
`MonthlySequence([1, 8, 15, 22, 29], excludes=[(2, 29)])`	Sequence recurring every 7 days each month, skipping the 29th Februa
`YearlySequence((12, 25))`	Sequence recurring every year on the 25th December
`Sequence.from_resource("ecmwf-4days")`	Pre-defined sequence (equivalent to `MonthlySequence(range(1, 30, 4), excludes=[(2, 29)])`)

`earthkit.time.climatology.model_climate_dates`(*reference: date, start: date | int, end: date | int, before: timedelta | int, after: timedelta | int, sequence: Sequence*)→ Iterator[date]

Generate a set of dates for a model climate

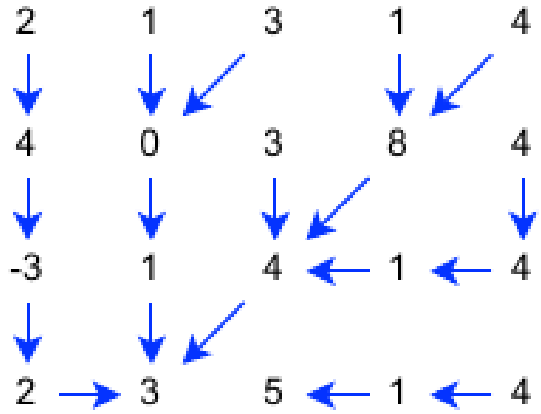
The set is created by combining yearly dates between `start` and `end`, for each date between `reference - before` and `reference + after`. If any of these dates is February 29th, the whole corresponing sequence will use February 28th instead.

Hydrology

earthkit hydro

- Hydrological computations
- River network objects
- Propagation of values
- Finding catchments

```
# conduct an accuflux
# this finds the amount of cells flowing into each point
accuflux_field = network.accuflux(unit_field)

# visualise the results
plot(accuflux_field)
```



Other components

- Previously known as **cascade**
- Contains an internal cascade engine for scheduling and executing task graphs

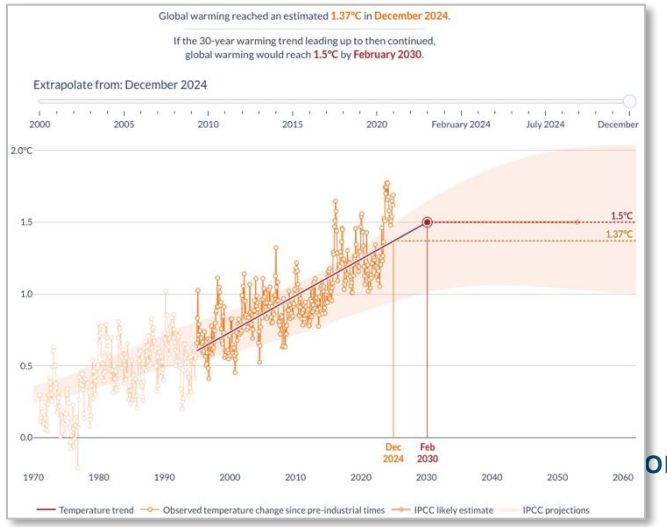
- Shared utilities between components

Examples usages today

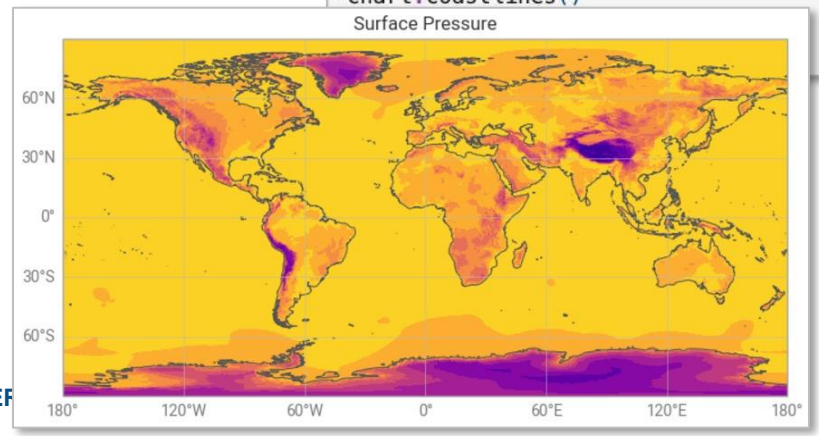
- The Anemoi machine learning framework (data handling and plotting)
- ECMWF's post-processing suite (data and time handling, thermodynamic computations)
- Recommended way to retrieve and visualise Destination Earth Digital Twin data via polytope
- Copernicus web applications, e.g. C3S Global Temperature Trend Monitor; on-the-fly aggregation of daily statistics; recommended way to access and process CDS datasets

AnemoI

```python
import earthkit.data
import earthkit.plots
import earthkit.regrid


request = {
    'activity': 'ScenarioMIP',
    'class': 'd1',
    'dataset': 'climate-dt',
    'date': '20200102',
    'experiment': 'SSP3-7.0',
    'expver': '0001',
    'generation': '1',
    'levtype': 'sfc',
    'model': 'IFS-NEMO',
    'param': '134/165/166',
    'realization': '1',
    'resolution': 'standard',
    'stream': 'clte',
    'time': '0100', # '0100/0200/0300/0400/0500/0600'
    'type': 'fc'
}

# data is an earthkit streaming object but with stream=False will
data = earthkit.data.from_source("polytope", "destination-earth",
```




```python
chart = earthkit.plots.Map(extent=[-180, 180, -90, 90])
chart.plot(
    data[0]
)

chart.title("Surface Pressure")
chart.coastlines()
```



Current / future work

earthkit regrid

- MIR-based interpolation using MIR (and the related SW stack) Python wheels.
- Brings: more grid types, subarea support, more interpolation types, missing value support

earthkit meteo

- Xarray input/output (currently flat data arrays only)

earthkit data

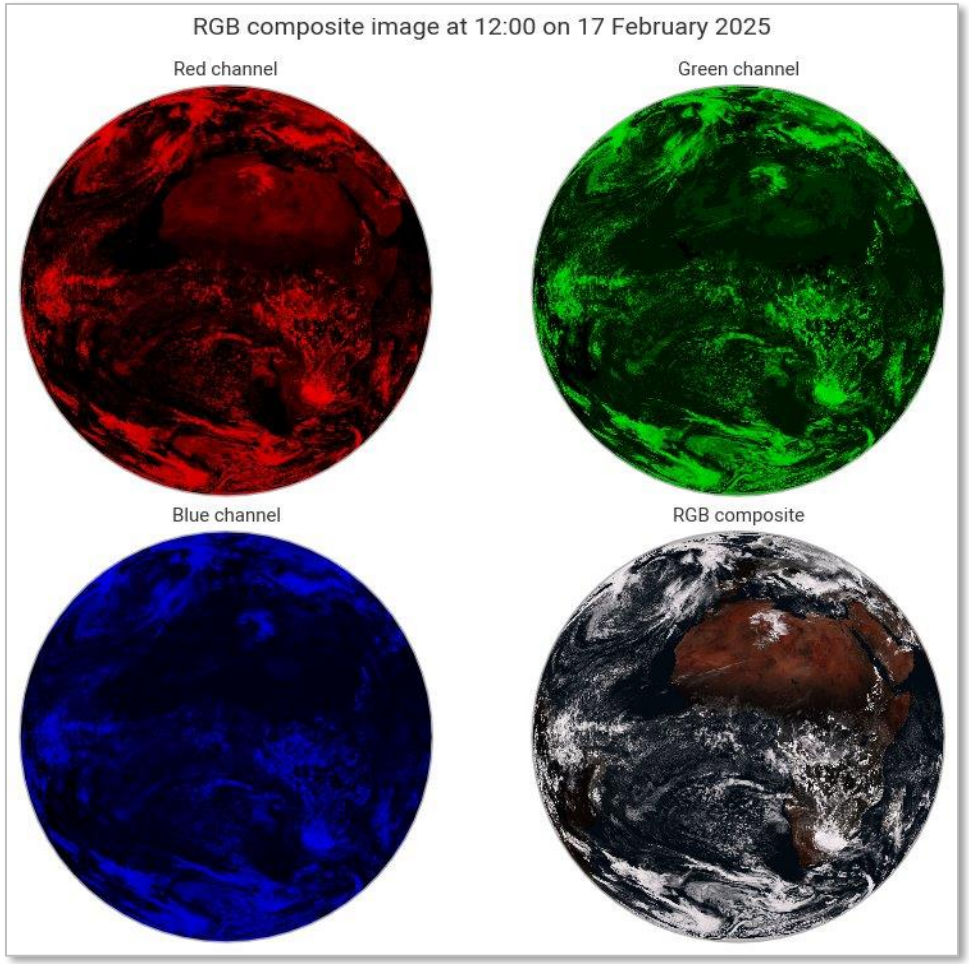
- New GRIB-to-xarray engine – will supersede cfgrib
- Will introduce maths operators on fields
- Working on new format-agnostic metadata engine in earthkit-data to enable work on GRIB and NetCDF without format-specific code

Current / future work

earthkit plots

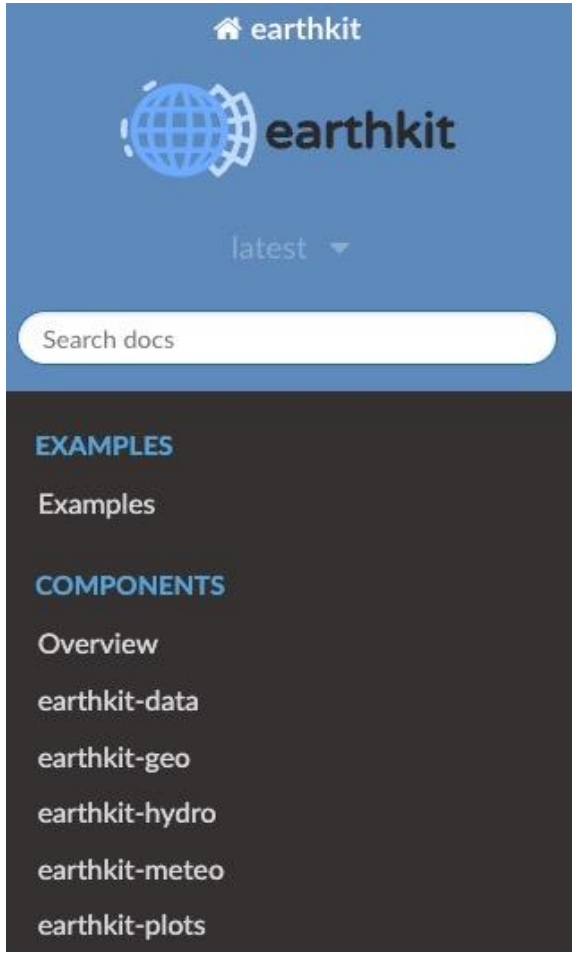
- New high-level time series function
  - Many more non-map plots will be supported
  - Flexible support for common data interfaces (xarray, numpy and pandas)
- New RGB composite plots

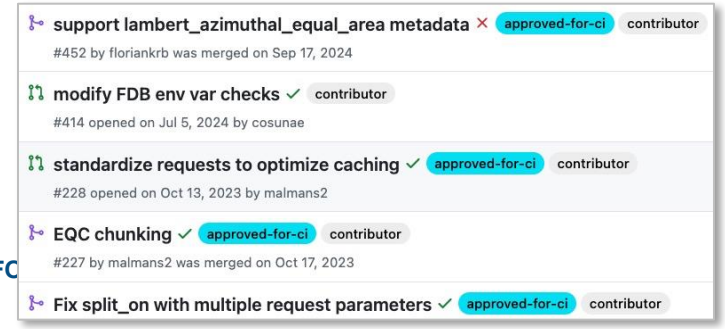
```
ek.plots.timeseries(
    ds,
    color="red",
    title="ERA5 hourly {variable_name} in {location:%c}, {location:%C} ({latitude:%Lt.1f}, {longitude:%Ln.1f})",
    units="celsius",
    mode="spline",
    xticks={
        "frequency": "D",
        "format": "%d %B",
        "period": True,
    },
).show()
```



Messages to take home

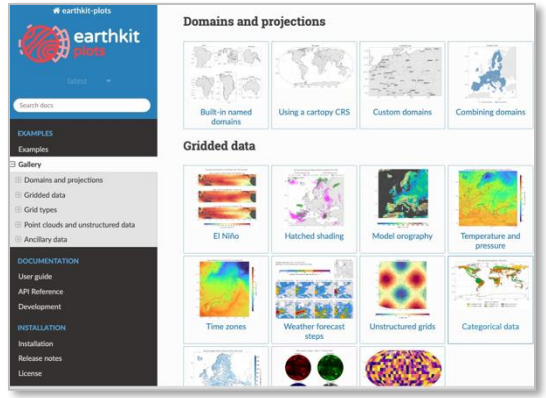
- ECMWF is leading development of a new set of Python packages for weather and climate science
- Suitable for operations and research
- All code is on GitHub for open development
  - e.g. https://github.com/ecmwf/earthkit-data
- Packages are on PyPi, e.g.
  - `pip install earthkit-data`
- Also an ‘umbrella’ package called ‘earthkit’
  - `pip install earthkit`
  - Installs the more mature components of earthkit, tested together
- Overall and package-specific documentation on Read The Docs:
  - https://earthkit.readthedocs.io/en/latest/
- Will ultimately replace Magics and Metview





Hands-on!

- Extra inspiration:
- https://earthkit.readthedocs.io/
- https://ecmwf.github.io/earthkit-website/



- Easiest: try the notebooks on binder:
- https://mybinder.org/v2/gh/ecmwf-training/2025-computing-training-week/main

- Alternatively try on your own machine from a virtualenv/conda env for example:
- `pip install earthkit==0.13.1 jupyter`
- `git clone https://github.com/ecmwf-training/2025-computing-training-week`
- `cd 2025-computing-training-week/earthkit`
- `jupyter notebook`

- For ECMWF users familiar with our JupyterHub:
- `git clone https://github.com/ecmwf-training/2025-computing-training-week # on ATOS`
- Start JupyterHub

- Also available on ATOS
- module load python3/new
- module load ecmwf-toolbox/new