xcube

Release 0.12.1

Brockmann Consult GmbH

Sep 21, 2022

GETTING STARTED

> **Warning:** This documentation is a work in progress and currently less than a

draft.

xcube has been developed to generate, manipulate, analyse, and publish data cubes from EO data.

OVERVIEW

xcube is an open-source Python package and toolkit that has been developed to provide Earth observation (EO) data in an analysis-ready form to users. xcube achieves this by carefully converting EO data sources into self-contained *data cubes* that can be published in the cloud.

1.1 Data Cube

The interpretation of the term *data cube* in the EO domain usually depends on the current context. It may refer to a data service such as Sentinel Hub, to some abstract API, or to a concrete set of spatial images that form a time-series.

This section briefly explains the specific concept of a data cube used in the xcube project - the *xcube dataset*.

1.2 xcube Dataset

1.2.1 Data Model

An xcube dataset contains one or more (geo-physical) data variables whose values are stored in cells of a common multi-dimensional, spatio-temporal grid. The dimensions are usually time, latitude, and longitude, however other dimensions may be present.

All xcube datasets are structured in the same way following a common data model. They are also self-describing by providing metadata for the cube and all cube's variables following the CF conventions. For details regarding the common data model, please refer to the *xcube Dataset Specification*.

A xcube dataset's in-memory representation in Python programs is an xarray.Dataset instance. Each dataset variable is represented by multi-dimensional xarray.DataArray that is arranged in non-overlapping, contiguous sub-regions called *data chunks*.

1.2.2 Data Chunks

Chunked variables allow for out-of-core computations of xcube dataset that don't fit in a single computer's RAM as data chunks can be processed independently from each other.

The way how dataset variables are sub-divided into smaller chunks - their *chunking* - has a substantial impact on processing performance and there is no single ideal chunking for all use cases. For time series analyses it is preferable to have chunks with a smaller spatial dimensions and larger time dimension, for spatial analyses and visualisation on using a map, the opposite is the case.

xcube provide tools for re-chunking of xcube datasets (*xcube chunk*, *xcube level*) and the xcube server (*xcube serve*) allows serving the same data cubes using different chunkings. For further reading have a look into the Chunking and Performance section of the xarray documentation.

1.2.3 Processing Model

When xcube datasets are opened, only the cube's structure and its metadata are loaded into memory. The actual data arrays of variables are loaded on-demand only, and only for chunks intersecting the desired sub-region.

Operations that generate new data variables from existing ones will be chunked in the same way. Therefore, such operation chains generate a processing graph providing a deferred, concurrent execution model.

1.2.4 Data Format

For the external, physical representation of xcube datasets we usually use the Zarr format. Zarr takes full advantage of data chunks and supports parallel processing of chunks that may originate from the local file system or from remote cloud storage such as S3 and GCS.

1.2.5 Python Packages

The xcube package builds heavily on Python's big data ecosystem for handling huge N-dimensional data arrays and exploiting cloud-based storage and processing resources. In particular, xcube's in-memory data model is provided by xarray, the memory management and processing model is provided through dask, and the external format is provided by zarr. xarray, dask, and zarr have increased their popularity for big data solutions over the last couple of years, for creating scalable and efficient EO data solutions.

1.3 Toolkit

On top of xarray, dask, zarr, and other popular Python data science packages, xcube provides various higher-level tools to generate, manipulate, and publish xcube datasets:

- *CLI* - access, generate, modify, and analyse xcube datasets using the `xcube` tool;
- *Python API* - access, generate, modify, and analyse xcube datasets via Python programs and notebooks;
- *Web API and Server* - access, analyse, visualize xcube datasets via an xcube server;
- *Viewer App* – publish and visualise xcube datasets using maps and time-series charts.

1.4 Workflows

The basic use case is to generate an xcube dataset and deploy it so that your users can access it:

1. generate an xcube dataset from some EO data sources using the *xcube gen* tool with a specific *input processor*.
2. optimize the generated xcube dataset with respect to specific use cases using the *xcube chunk* tool.
3. optimize the generated xcube dataset by consolidating metadata and elimination of empty chunks using *xcube optimize* and *xcube prune* tools.
4. deploy the optimized xcube dataset(s) to some location (e.g. on AWS S3) where users can access them.

Then you can:

5. access, analyse, modify, transform, visualise the data using the *Python API* and xarray API through Python programs or JupyterLab, or

6. extract data points by coordinates from a cube using the *xcube extract* tool, or

7. resample the cube in time to generate temporal aggregations using the *xcube resample* tool.

Another way to provide the data to users is via the *xcube server*, that provides a RESTful API and a WMTS. The latter is used to visualise spatial subsets of xcube datasets efficiently at any zoom level. To provide optimal visualisation and data extraction performance through the xcube server, xcube datasets may be prepared beforehand. Steps 8 to 10 are optional.

8. verify a dataset to be published conforms with the *xcube Dataset Specification* using the *xcube verify* tool.

9. adjust your dataset chunking to be optimal for generating spatial image tiles and generate a multi-resolution image pyramid using the *xcube chunk* and *xcube level* tools.

10. create a dataset variant optimal for time series-extraction again using the *xcube chunk* tool.

11. configure xcube datasets and publish them through the xcube server using the *xcube serve* tool.

You may then use a WMTS-compatible client to visualise the datasets or develop your own xcube server client that will make use of the xcube's REST API.

The easiest way to visualise your data is using the xcube *Viewer App*, a single-page web application that can be configured to work with xcube server URLs.

EXAMPLES

When you follow the examples section you can build your first tiny xcube dataset and view it in the xcube-viewer by using the xcube server. The examples section is still growing and improving :)

Have fun exploring xcube!

> **Warning:** This chapter is a work in progress and currently less than a draft.

2.1 Generating an xcube dataset

In the following example a tiny demo xcube dataset is generated.

2.1.1 Analysed Sea Surface Temperature over the Global Ocean

Input data for this example is located in the xcube repository. The input files contain analysed sea surface temperature and sea surface temperature anomaly over the global ocean and are provided by Copernicus Marine Environment Monitoring Service. The data is described in a dedicated Product User Manual.

Before starting the example, you need to activate the xcube environment:

```
$ conda activate xcube
```

If you want to take a look at the input data you can use cli/xcube dump to print out the metadata of a selected input file:

```
$ xcube dump examples/gen/data/20170605120000-UKMO-L4_GHRSST-SSTfnd-OSTIAanom-GLOB-v02.0-
↪fv02.0.nc
```

```
<xarray.Dataset>
Dimensions:      (lat: 720, lon: 1440, time: 1)
Coordinates:
  * lat          (lat) float32 -89.875 -89.625 -89.375 ... 89.375 89.625 89.875
  * lon          (lon) float32 0.125 0.375 0.625 ... 359.375 359.625 359.875
  * time         (time) object 2017-06-05 12:00:00
Data variables:
    sst_anomaly  (time, lat, lon) float32 ...
    analysed_sst (time, lat, lon) float32 ...
Attributes:
    Conventions:               CF-1.4
```

(continues on next page)

(continued from previous page)

```
        title:                     Global SST & Sea Ice Anomaly, L4 OSTIA, 0.25 ...
        summary:                   A merged, multi-sensor L4 Foundation SST anom...
        references:                Donlon, C.J., Martin, M., Stark, J.D., Robert...
        institution:               UKMO
        history:                   Created from sst:temperature regridded with a...
        comment:                   WARNING Some applications are unable to prope...
        license:                   These data are available free of charge under...
        id:                        UKMO-L4LRfnd_GLOB-OSTIAanom
        naming_authority:          org.ghrsst
        product_version:           2.0
        uuid:                      5c1665b7-06e8-499d-a281-857dcbfd07e2
        gds_version_id:            2.0
        netcdf_version_id:         3.6
        date_created:              20170606T061737Z
        start_time:                20170605T000000Z
        time_coverage_start:       20170605T000000Z
        stop_time:                 20170606T000000Z
        time_coverage_end:         20170606T000000Z
        file_quality_level:        3
        source:                    UKMO-L4HRfnd-GLOB-OSTIA
        platform:                  Aqua, Envisat, NOAA-18, NOAA-19, MetOpA, MSG1...
        sensor:                    AATSR, AMSR, AVHRR, AVHRR_GAC, SEVIRI, TMI
        metadata_conventions:      Unidata Observation Dataset v1.0
        metadata_link:             http://data.nodc.noaa.gov/NESDIS_DataCenters/...
        keywords:                  Oceans > Ocean Temperature > Sea Surface Temp...
        keywords_vocabulary:       NASA Global Change Master Directory (GCMD) Sc...
        standard_name_vocabulary:  NetCDF Climate and Forecast (CF) Metadata Con...
        westernmost_longitude:     0.0
        easternmost_longitude:     360.0
        southernmost_latitude:     -90.0
        northernmost_latitude:     90.0
        spatial_resolution:        0.25 degree
        geospatial_lat_units:      degrees_north
        geospatial_lat_resolution: 0.25 degree
        geospatial_lon_units:      degrees_east
        geospatial_lon_resolution: 0.25 degree
        acknowledgment:            Please acknowledge the use of these data with...
        creator_name:              Met Office as part of CMEMS
        creator_email:             servicedesk.cmems@mercator-ocean.eu
        creator_url:               http://marine.copernicus.eu/
        project:                   Group for High Resolution Sea Surface Tempera...
        publisher_name:            GHRSST Project Office
        publisher_url:             http://www.ghrsst.org
        publisher_email:           ghrsst-po@nceo.ac.uk
        processing_level:          L4
        cdm_data_type:             grid
```

Below an example xcube dataset will be created, which will contain the variable analysed_sst. The metadata for a specific variable can be viewed by:

```
$ xcube dump examples/gen/data/20170605120000-UKMO-L4_GHRSST-SSTfnd-OSTIAanom-GLOB-v02.0-
↪fv02.0.nc --var analysed_sst
```

```
<xarray.DataArray 'analysed_sst' (time: 1, lat: 720, lon: 1440)>
[1036800 values with dtype=float32]
Coordinates:
  * lat       (lat) float32 -89.875 -89.625 -89.375 ... 89.375 89.625 89.875
  * lon       (lon) float32 0.125 0.375 0.625 0.875 ... 359.375 359.625 359.875
  * time      (time) object 2017-06-05 12:00:00
Attributes:
    long_name:      analysed sea surface temperature
    standard_name:  sea_surface_foundation_temperature
    type:           foundation
    units:          kelvin
    valid_min:      -300
    valid_max:      4500
    source:         UKMO-L4HRfnd-GLOB-OSTIA
    comment:
```

For creating a toy xcube dataset you can execute the command-line below. Please adjust the paths to your needs:

```
$ xcube gen -o "your/output/path/demo_SST_xcube.zarr" -c examples/gen/config_files/xcube_
↪sst_demo_config.yml --sort examples/gen/data/*.nc
```

The configuration file specifies the input processor, which in this case is the default one. The output size is 10240, 5632. The bounding box of the data cube is given by `output_region` in the configuration file. The output format (`output_writer_name`) is defined as well. The chunking of the dimensions can be set by the `chunksizes` attribute of the `output_writer_params` parameter, and in the example configuration file the chunking is set for latitude and longitude. If the chunking is not set, a automatic chunking is applied. The spatial resampling method (`output_resampling`) is set to 'nearest' and the configuration file contains only one variable which will be included into the xcube dataset - 'analysed-sst'.

The Analysed Sea Surface Temperature data set contains the variable already as needed. This means no pixel masking needs to be applied. However, this might differ depending on the input data. You can take a look at a configuration file which takes Sentinel-3 Ocean and Land Colour Instrument (OLCI) as input files, which is a bit more complex. The advantage of using pixel expressions is, that the generated cube contains only valid pixels and the user of the data cube does not have to worry about something like land-masking or invalid values. Furthermore, the generated data cube is spatially regular. This means the data are aligned on a common spatial grid and cover the same region. The time stamps are kept from the input data set.

Caution: If you have input data that has file names not only varying with the time stamp but with e.g. A and B as well, you need to pass the input files in the desired order via a text file. Each line of the text file should contain the path to one input file. If you pass the input files in a desired order, then do not use the parameter `--sort` within the commandline interface.

2.1.2 Optimizing and pruning a xcube dataset

If you want to optimize your generated xcube dataset e.g. for publishing it in a xcube viewer via xcube serve you can use cli/xcube optimize:

```
$ xcube optimize demo_SST_xcube.zarr -C
```

By executing the command above, an optimized xcube dataset called demo_SST_xcube-optimized.zarr will be created. You can take a look into the directory of the original xcube dataset and the optimized one, and you will notice that a file called .zmetadata. .zmetadata contains the information stored in .zattrs and .zarray of each variable of the xcube dataset and makes requests of metadata faster. The option `-C` optimizes coordinate variables by converting any chunked arrays into single, non-chunked, contiguous arrays.

For deleting empty chunks cli/xcube prune can be used. It deletes all data files associated with empty (NaN-only) chunks of an xcube dataset, and is restricted to the ZARR format.

```
$ xcube prune demo_SST_xcube-optimized.zarr
```

The pruned xcube dataset is saved in place and does not need an output path. The size of the xcube dataset was 6,8 MB before pruning it and 6,5 MB afterwards. According to the output printed to the terminal, 30 block files were deleted.

The metadata of the xcube dataset can be viewed with cli/xcube dump as well:

```
$ xcube dump demo_SST_xcube-optimized.zarr
```

```
<xarray.Dataset>
Dimensions:        (bnds: 2, lat: 5632, lon: 10240, time: 3)
Coordinates:
  * lat            (lat) float64 62.67 62.66 62.66 62.66 ... 48.01 48.0 48.0
    lat_bnds       (lat, bnds) float64 dask.array<shape=(5632, 2), chunksize=(5632, 2)>
  * lon            (lon) float64 -16.0 -16.0 -15.99 -15.99 ... 10.66 10.66 10.67
    lon_bnds       (lon, bnds) float64 dask.array<shape=(10240, 2), chunksize=(10240, 2)>
  * time           (time) datetime64[ns] 2017-06-05T12:00:00 ... 2017-06-07T12:00:00
    time_bnds      (time, bnds) datetime64[ns] dask.array<shape=(3, 2), chunksize=(3, 2)>
Dimensions without coordinates: bnds
Data variables:
    analysed_sst  (time, lat, lon) float64 dask.array<shape=(3, 5632, 10240),
→chunksize=(1, 704, 640)>
Attributes:
    acknowledgment:            Data Cube produced based on data provided by ...
    comment:
    contributor_name:
    contributor_role:
    creator_email:             info@brockmann-consult.de
    creator_name:              Brockmann Consult GmbH
    creator_url:               https://www.brockmann-consult.de
    date_modified:             2019-09-25T08:50:32.169031
    geospatial_lat_max:        62.666666666666664
    geospatial_lat_min:        48.0
    geospatial_lat_resolution: 0.002604166666666666
    geospatial_lat_units:      degrees_north
    geospatial_lon_max:        10.666666666666664
    geospatial_lon_min:        -16.0
    geospatial_lon_resolution: 0.0026041666666666665
    geospatial_lon_units:      degrees_east
    history:                   xcube/reproj-snap-nc
    id:                        demo-bc-sst-sns-l2c-v1
    institution:               Brockmann Consult GmbH
    keywords:
    license:                   terms and conditions of the DCS4COP data dist...
    naming_authority:          bc
    processing_level:          L2C
    project:                   xcube
    publisher_email:           info@brockmann-consult.de
    publisher_name:            Brockmann Consult GmbH
    publisher_url:             https://www.brockmann-consult.de
    references:                https://dcs4cop.eu/
```

(continues on next page)

```
    source:                  CMEMS Global SST & Sea Ice Anomaly Data Cube
    standard_name_vocabulary:
    summary:
    time_coverage_end:       2017-06-08T00:00:00.000000000
    time_coverage_start:     2017-06-05T00:00:00.000000000
    title:                   CMEMS Global SST Anomaly Data Cube
```

The metadata for the variable analysed_sst can be viewed:

```
$ xcube dump demo_SST_xcube-optimized.zarr --var analysed_sst
```

```
<xarray.DataArray 'analysed_sst' (time: 3, lat: 5632, lon: 10240)>
dask.array<shape=(3, 5632, 10240), dtype=float64, chunksize=(1, 704, 640)>
Coordinates:
  * lat      (lat) float64 62.67 62.66 62.66 62.66 ... 48.01 48.01 48.0 48.0
  * lon      (lon) float64 -16.0 -16.0 -15.99 -15.99 ... 10.66 10.66 10.66 10.67
  * time     (time) datetime64[ns] 2017-06-05T12:00:00 ... 2017-06-07T12:00:00
Attributes:
    comment:
    long_name:            analysed sea surface temperature
    source:               UKMO-L4HRfnd-GLOB-OSTIA
    spatial_resampling:   Nearest
    standard_name:        sea_surface_foundation_temperature
    type:                 foundation
    units:                kelvin
    valid_max:            4500
    valid_min:            -300
```

> **Warning:** This chapter is a work in progress and currently less than a draft.

2.2 Publishing xcube datasets

This example demonstrates how to run an xcube server to publish existing xcube datasets.

2.2.1 Running the server

To run the server on default port 8080 using the demo configuration:

```
$ xcube serve --verbose -c examples/serve/demo/config.yml
```

To run the server using a particular xcube dataset path and styling information for a variable:

```
$ xcube serve --styles conc_chl=(0,20,"viridis") examples/serve/demo/cube-1-250-250.zarr
```

2.2.2 Test it

After starting the server, check the various functions provided by xcube Web API.

- **Datasets:**
  - Get datasets
  - Get dataset details
  - Get dataset coordinates
- **Color bars:**
  - Get color bars
  - Get color bars (HTML)
- **WMTS:**
  - Get WMTS KVP Capabilities (XML)
  - Get WMTS KVP local tile (PNG)
  - Get WMTS KVP remote tile (PNG)
  - Get WMTS REST Capabilities (XML)
  - Get WMTS REST local tile (PNG)
  - Get WMTS REST remote tile (PNG)
- **Tiles**
  - Get tile (PNG)
  - Get tile grid for OpenLayers 4.x
  - Get tile grid for Cesium 1.x
  - Get legend for layer (PNG)
- **Time series service (preliminary & unstable, will likely change soon)**
  - Get time stamps per dataset
  - Get time series for single point
- **Places service (preliminary & unstable, will likely change soon>`_**
  - Get all features
  - Get all features of collection "inside-cube"
  - Get all features for dataset "local"
  - Get all features of collection "inside-cube" for dataset "local"

2.2.3 xcube Viewer

xcube datasets published through `xcube serve` can be visualised using the xcube-viewer web application. To do so, run `xcube serve` with the `--show` flag.

In order make this option usable, xcube-viewer must be installed and build:

1. Download and install yarn.

2. Download and build xcube-viewer:

```
$ git clone https://github.com/dcs4cop/xcube-viewer.git
$ cd xcube-viewer
$ yarn build
```

3. Configure `xcube serve` so it finds the xcube-viewer On Linux (please adjust path):

```
 $ export XCUBE_VIEWER_PATH=/abs/path/to/xcube-viewer/build
```

On Windows (please adjust path):

```
> SET XCUBE_VIEWER_PATH=/abs/path/to/xcube-viewer/build
```

4. Then run `xcube serve --show`:

```
$ xcube serve --show --styles conc_chl=(0,20,"viridis") examples/serve/demo/cube-1-250-
→250.zarr
```

Viewing the generated xcube dataset described in the example *Generating an xcube dataset*:

```
$ xcube serve --show --styles "analysed_sst=(280,290,'plasma')" demo_SST_xcube-optimized.
→zarr
```
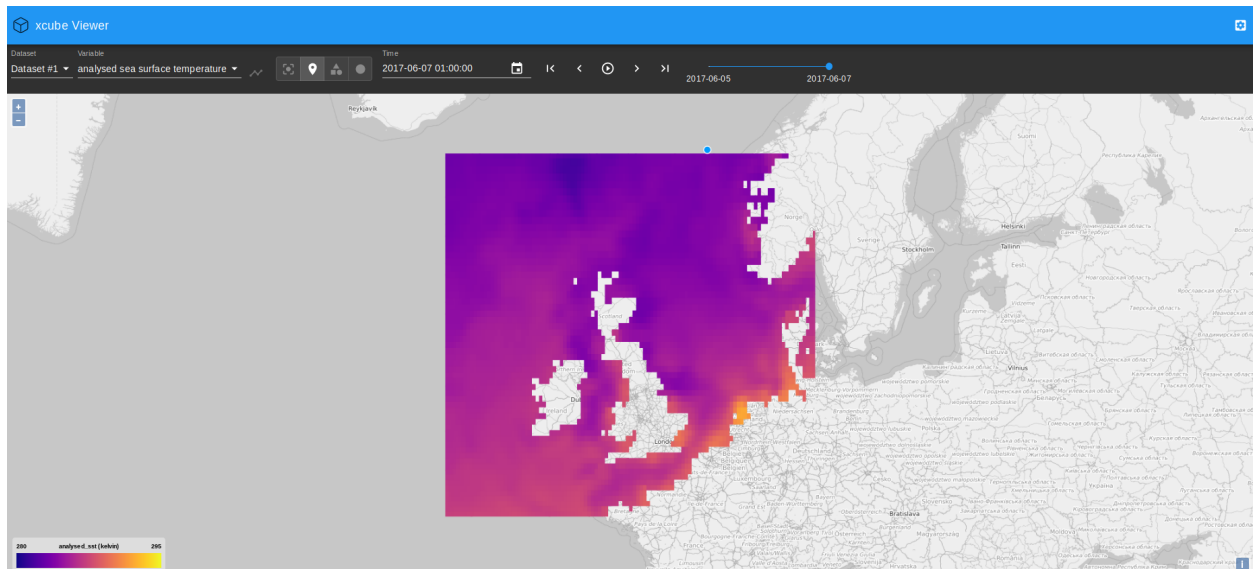

In case you get an error message "cannot reach server" on the very bottom of the web app's main window, refresh the page.

You can play around with the value range displayed in the viewer, here it is set to min=280K and max=290K. The colormap used for mapping can be modified as well and the colormaps provided by matplotlib can be used.

2.2.4 Other clients

There are example HTML pages for some tile server clients. They need to be run in a web server. If you don't have one, you can use Node's `httpserver`:

```
$ npm install -g httpserver
```

After starting both the xcube server and web server, e.g. on port 9090:

```
$ httpserver -d -p 9090
```

you can run the client demos by following their links given below.

OpenLayers

- OpenLayers 4 Demo
- OpenLayers 4 Demo with WMTS

Cesium

To run the Cesium Demo first download Cesium and unpack the zip into the `xcube serve` source directory so that there exists an `./Cesium-x.y.z` sub-directory. You may have to adapt the Cesium version number in the demo's HTML file.

INSTALLATION

xcube can be installed from a released conda package, or directly from a copy of the source code repository.

The first two sections below give instructions for installation using conda, available as part of the miniconda distribution. If installation using conda proves to be unacceptably slow, mamba can be used instead (see *Installation using mamba*).

3.1 Installation from the conda package

Into a currently active, existing conda environment (>= Python 3.7)

```
$ conda install -c conda-forge xcube
```

Into a new conda environment named `xcube`:

```
$ conda create -c conda-forge -n xcube xcube
```

The argument to the `-n` option can be changed to create a differently named environment.

3.2 Installation from the source code repository

First, clone the repository and create a conda environment from it:

```
$ git clone https://github.com/dcs4cop/xcube.git
$ cd xcube
$ conda env create
```

From this point on, all instructions assume that your current directory is the root of the xcube repository.

The `conda env create` command above creates an environment according to the specifications in the `environment.yml` file in the repository, which by default takes the name `xcube`. Then, to activate the environment and install xcube from the repository:

```
$ conda activate xcube
$ pip install --no-deps --editable .
```

The second command installs xcube in 'editable mode', meaning that it will be run directly from the repository, and changes to the code in the repository will take immediate effect without reinstallation. (As an alternative to pip, the command `python setup.py develop` can be used, but this is no longer recommended. Among other things, `pip` has the advantage of allowing easy deinstallation of installed packages.)

To update the install to the latest repository version and update the environment to reflect to any changes in `environment.yml`:

```
$ conda activate xcube
$ git pull --force
$ conda env update -n xcube --file environment.yml --prune
```

To install `pytest` and run the unit test suite:

```
$ conda install pytest
$ pytest
```

To analyse test coverage (after installing pytest as above):

```
$ pytest --cov=xcube
```

To produce an HTML coverage report:

```
$ pytest --cov-report html --cov=xcube
```

3.3 Installation using mamba

Mamba is a dramatically faster drop-in replacement for the conda tool. Mamba itself can be installed using conda. If installation using conda proves to be unacceptably slow, it is recommended to install mamba, as follows:

```
$ conda create -n xcube python=3.8
$ conda activate xcube
$ conda install -c conda-forge mamba
```

This creates a conda environment called `xcube`, activates the environment, and installs mamba in it. To install xcube from its conda-forge package, you can now use:

```
$ mamba install -c conda-forge xcube
```

Alternatively, to install xcube directly from the repository:

```
$ git clone https://github.com/dcs4cop/xcube.git
$ cd xcube
$ mamba env create
$ pip install --no-deps --editable .
```

3.4 Docker

To start a demo using docker use the following commands

```
$ docker build -t [your name] .
$ docker run [your name]
```

$ docker run -d -p [host port]:8080 [your name]

Example 1:

```
$  docker build -t xcube:0.10.0 .
$  docker run xcube:0.10.0
```

This will create the docker container and list the functionality of the `xcube` cli.

Example 2:

```
$  docker build -t xcube:0.10.0 .
$  docker run -d -p 8001:8080 "xcube:0.10.0 xcube serve -v --address 0.0.0.0 --port 8080␣
↪-c /home/xcube/examples/serve/demo/config.yml"
$  docker ps
```

This will have started a service in the background which can be accessed through port 8001, as the startup of a service is configured as default behaviour.

CLI

The xcube command-line interface (CLI) is a single executable cli/xcube with several sub-commands comprising functions ranging from xcube dataset generation, over analysis and manipulation, to dataset publication.

4.1 Common Arguments and Options

Most of the commands operate on inputs that are xcube datasets. Such inputs are consistently named `CUBE` and provided as one or more command arguments. CUBE inputs may be a path into the local file system or a path into some object storage bucket, e.g. in AWS S3. Command inputs of other types are consistently called `INPUT`.

Many commands also output something, i.e. are writing files. The paths or names of such outputs are consistently provided by the `-o OUTPUT` or `--output OUTPUT` option. As the output is an option, there is usually a default value for it. If multiply file formats are supported, commands usually provide a `-f FORMAT` or `--format FORMAT` option. If omitted, the format may be guessed from the output's name.

4.2 Cube generation

4.2.1 `xcube gen`

Synopsis

Generate xcube dataset.

```
$ xcube gen --help
```

```
Usage: xcube gen [OPTIONS] [INPUT]...

  Generate xcube dataset. Data cubes may be created in one go or
  successively for all given inputs. Each input is expected to provide a
  single time slice which may be appended, inserted or which may replace an
  existing time slice in the output dataset. The input paths may be one or
  more input files or a pattern that may contain wildcards '?', '*', and
  '**'. The input paths can also be passed as lines of a text file. To do
  so, provide exactly one input file with ".txt" extension which contains
  the actual input paths to be used.

Options:
  -P, --proc INPUT-PROCESSOR      Input processor name. The available input
```

(continued from previous page)

	processor names **and** additional information about `input` processors can be accessed by calling xcube gen --info . Defaults to `"default"`, an `input` processor that can deal **with** simple datasets whose variables have dimensions (`"lat"`, `"lon"`) **and** conform **with** the CF conventions.
`-c, --config CONFIG`	xcube dataset configuration file **in** YAML `format`. More than one config `input` file **is** allowed.When passing several config files, they are merged considering the order passed via command line.
`-o, --output OUTPUT`	Output path. Defaults to `'out.zarr'`
`-f, --format FORMAT`	Output `format`. Information about output formats can be accessed by calling xcube gen --info. If omitted, the `format` will be guessed **from the** given output path.
`-S, --size SIZE`	Output size **in** pixels using `format` `"<width>,<height>"`.
`-R, --region REGION`	Output region using `format "<lon-min>,<lat-min>,<lon-max>,<lat-max>"`
`--variables, --vars VARIABLES`	Variables to be included **in** output. Comma-separated `list` of names which may contain wildcard characters `"*"` **and** `"?"`.
`--resampling␣` ↪`[Average|Bilinear|Cubic|CubicSpline|Lanczos|Max|Median|Min|Mode|Nearest|Q1|Q3]`	Fallback spatial resampling algorithm to be used **for** all variables. Defaults to `'Nearest'`. The choices **for** the resampling algorithm are: [`'Average'`, `'Bilinear'`, `'Cubic'`, `'CubicSpline'`, `'Lanczos'`, `'Max'`, `'Median'`, `'Min'`, `'Mode'`, `'Nearest'`, `'Q1'`, `'Q3'`]
`-a, --append`	Deprecated. The command will now always create, insert, replace, **or** append `input` slices.
`--prof`	Collect profiling information **and** dump results after processing.
`--no_sort`	The `input` file `list` will **not** be `sorted` before creating the xcube dataset. If `--no_sort` parameter **is** passed, the order of the `input list` will be kept. This parameter should be used **for** better performance, provided that the `input` file `list` **is in** correct order (continuous time).
`-I, --info`	Displays additional information about `format` options **or** about `input` processors.
`--dry_run`	Just read **and** process inputs, but don`'t` produce `any` outputs.
`--help`	Show this message **and** exit.

Below is the ouput of a `xcube gen --info` call showing five input processors installed via plugins.

```
$ xcube gen --info
```

```
input processors to be used with option --proc:
  default                       Single-scene NetCDF/CF inputs in xcube standard␣
→format
  rbins-seviri-highroc-scene-l2     RBINS SEVIRI HIGHROC single-scene Level-2 NetCDF␣
→inputs
  rbins-seviri-highroc-daily-l2     RBINS SEVIRI HIGHROC daily Level-2 NetCDF inputs
  snap-olci-highroc-l2          SNAP Sentinel-3 OLCI HIGHROC Level-2 NetCDF inputs
  snap-olci-cyanoalert-l2       SNAP Sentinel-3 OLCI CyanoAlert Level-2 NetCDF inputs
  vito-s2plus-l2                VITO Sentinel-2 Plus Level 2 NetCDF inputs

For more input processors use existing "xcube-gen-..." plugins from the github␣
→organisation DCS4COP or write own plugin.


output formats to be used with option --format:
  csv                 (*.csv)      CSV file format
  mem                 (*.mem)      In-memory dataset I/O
  netcdf4             (*.nc)       NetCDF-4 file format
  zarr                (*.zarr)     Zarr file format (http://zarr.readthedocs.io)
```

Configuration File

Configuration files passed to `xcube gen` via the `-c, --config` option use YAML format. Multiple configuration files may be given. In this case all configurations are merged into a single one. Parameter values will be overwritten by subsequent configurations if they are scalars. If they are objects / mappings, their values will be deeply merged.

The following parameters can be used in the configuration files:

input_processor [str] The name of an *input processor*. See `-P, --proc` option above.

> **Default** The default value is `'default'`, xcube's default input processor. It can ingest and process inputs that
>
> > - use an `EPSG:4326` (or compatible) grid;
> >
> > - have 1-D `lon` and `lat` coordinate variables using WGS84 coordinates and decimal degrees;
> >
> > - have a decodable 1-D `time` coordinate or define the one of the following global attribute pairs `time_coverage_start` and `time_coverage_end`, `time_start` and `time_end` or `time_stop`;
> >
> > - provide data variables with the dimensions `time`, `lat`, `lon`, in this order.
> >
> > - conform to the `` `CF Conventions` ``_.

output_size [[int, int]] The spatial dimension sizes of the output dataset given as number of grid cells in longitude and latitude direction (width and height).

output_region [[float, float, float, float]] The spatial extent of output datasets given as a bounding box [lat-min, lat-min, lon-max, lat-max] using decimal degrees.

output_variables [[*variable-definitions*]] The definition of variables that will be included in the output dataset. Each variable definition may be just a name or a mapping from a name to variable attributes. If it is just a name it must be the name of an existing variable either in the INPUT or in `processed_variables`. If the variable

definition is a mapping, some of the attributes affect the way how variables are processed. All but the `name` attributes become variable metadata in the output.

name [str] The new name of the variable in the output.

valid_pixel_expression [str] An expression used to mask this variable, see *Expressions*. The expression identifies all valid pixels in each INPUT.

resampling [str] The resampling method used. See `--resampling` option above.

Default By default, all variables in INPUT will occur in output.

processed_variables [[*variable-definitions*]] The definition of variables that will be produced or processed after reading each INPUT. The main purpose is to generate intermediate variables that can be referred to in the `expression` in other variable definitions in `processed_variables` and `valid_pixel_expression` in variable definitions in `output_variables`. The following attributes are recognised:

expression [str] An expression used to produce this variable, see *Expressions*.

output_writer_name [str] The name of a supported output format. May be one of `'zarr'`, `'netcdf4'`, `'mem'`.

Default `'zarr'`

output_writer_params [str] A mapping that defines parameters that are passed to output writer denoted by `output_writer_name`. Through the `output_writer_params` a packing of the variables may be defined. If not specified the default does not apply any packing which results in:

```
_FillValue:   nan
dtype:        dtype('float32')
```

and for coordinate variables

```
dtype:        dtype('int64')
```

The user may specify a different packing variables, which might be useful for reducing the storage size of the datacubes. Currently it is only implemented for zarr format. This may be done by passing the parameters for packing as the following:

```
output_writer_params:

  packing:
    analysed_sst:
      scale_factor: 0.07324442274239326
      add_offset: -300.0
      dtype: 'uint16'
      _FillValue: 0.65535
```

Furthermore the compressor may be defined as well by, if not specified the default compressor (cname='lz4', clevel=5, shuffle=SHUFFLE, blocksize=0) is used.

```
output_writer_params:

  compressor:
    cname: 'zstd'
    clevel: 1
    shuffle: 2
```

output_metadata [[*attribute-definitions*]] General metadata that will be present in the output dataset as global attributes. You can put any common CF attributes here.

Any attributes that are mappings will be "flattened" by concatenating the attribute names using the underscrore character. For example,:

```
publisher:
  name:  "Brockmann Consult GmbH"
  url:   "https://www.brockmann-consult.de"
```

will create the two entries:

```
publisher_name:  "Brockmann Consult GmbH"
publisher_url:   "https://www.brockmann-consult.de"
```

Expressions

Expressions are plain text values of the `expression` and `valid_pixel_expression` attributes of the variable definitions in the `processed_variables` and `output_variables` parameters. The expression syntax is that of standard Python. `xcube gen` uses expressions to produce new variables listed in `processed_variables` and to mask variables by the `valid_pixel_expression`.

An expression may refer any variables in the INPUT datasets and any variables defined by the `processed_variables` parameter. Expressions may make use of most of the standard Python operators and may apply all numpy ufuncs to referred variables. Also most of the xarray.DataArray API may be used on variables within an expression.

In order to utilise flagged variables, the syntax `variable_name.flag_name` can be used in expressions. According to the CF Conventions, flagged variables are variables whose metadata include the attributes `flag_meanings` and `flag_values` and/or `flag_masks`. The `flag_meanings` attribute enumerates the allowed values for `flag_name`. The flag attributes must be present in the variables of each INPUT.

Example

An example that uses a configuration file only:

```
$ xcube gen --config ./config.yml /data/eo-data/SST/2018/**/*.nc
```

An example that uses the default input processor and passes all other configuration via command-line options:

```
$ xcube gen -S 2000,1000 -R 0,50,5,52.5 --vars conc_chl,conc_tsm,kd489,c2rcc_flags,
→quality_flags -o hiroc-cube.zarr /data/eo-data/SST/2018/**/*.nc
```

Some input processors have been developed for specific EO data sources used within the DCS4COP project. They may serve as examples how to develop input processor plug-ins:

- xcube-gen-rbins
- xcube-gen-bc
- xcube-gen-vito

Python API

The related Python API function is `xcube.core.gen.gen.gen_cube()`.

4.2.2 `xcube grid`

> **Attention:** This tool will likely change in the near future.

Synopsis

Find spatial xcube dataset resolutions and adjust bounding boxes.

```
$ xcube grid --help
```

```
Usage: xcube grid [OPTIONS] COMMAND [ARGS]...

  Find spatial xcube dataset resolutions and adjust bounding boxes.

  We find suitable resolutions with respect to a possibly regional fixed
  Earth grid and adjust regional spatial bounding boxes to that grid. We
  also try to select the resolutions such that they are taken from a certain
  level of a multi-resolution pyramid whose level resolutions increase by a
  factor of two.

  The graticule at a given resolution level L within the grid is given by

      RES(L) = COVERAGE * HEIGHT(L)
      HEIGHT(L) = HEIGHT_0 * 2 ^ L
      LON(L, I) = LON_MIN + I * HEIGHT_0 * RES(L)
      LAT(L, J) = LAT_MIN + J * HEIGHT_0 * RES(L)

  With

      RES:      Grid resolution in degrees.
      HEIGHT:   Number of vertical grid cells for given level
      HEIGHT_0: Number of vertical grid cells at lowest resolution level.

  Let WIDTH and HEIGHT be the number of horizontal and vertical grid cells
  of a global grid at a certain LEVEL with WIDTH * RES = 360 and HEIGHT *
  RES = 180, then we also force HEIGHT = TILE * 2 ^ LEVEL.

Options:
  --help  Show this message and exit.

Commands:
  abox    Adjust a bounding box to a fixed Earth grid.
  levels  List levels for a resolution or a tile size.
  res     List resolutions close to a target resolution.
```

Example: Find suitable target resolution for a ~300m (Sentinel 3 OLCI FR resolution) fixed Earth grid within a deviation of 5%.

```
$ xcube grid res 300m -D 5%
```

```
TILE    LEVEL   HEIGHT  INV_RES RES (deg)           RES (m), DELTA_RES (%)
540     7       69120   384     0.0026041666666666665   289.9   -3.4
4140    4       66240   368     0.002717391304347826    302.5   0.8
8100    3       64800   360     0.002777777777777778    309.2   3.1
...
```

289.9m is close enough and provides 7 resolution levels, which is good. Its inverse resolution is 384, which is the fixed Earth grid identifier.

We want to see if the resolution pyramid also supports a resolution close to 10m (Sentinel 2 MSI resolution).

```
$ xcube grid levels 384 -m 6
```

```
LEVEL   HEIGHT  INV_RES RES (deg)           RES (m)
0       540     3       0.3333333333333333      37106.5
1       1080    6       0.16666666666666666     18553.2
2       2160    12      0.08333333333333333     9276.6
...
11      1105920 6144    0.00016276041666666666  18.1
12      2211840 12288   8.138020833333333e-05   9.1
13      4423680 24576   4.0690104166666664e-05  4.5
```

This indicates we have a resolution of 9.1m at level 12.

Lets assume we have xcube dataset region with longitude from 0 to 5 degrees and latitudes from 50 to 52.5 degrees. What is the adjusted bounding box on a fixed Earth grid with the inverse resolution 384?

```
$ xcube grid abox  0,50,5,52.5  384
```

```
Orig. box coord. = 0.0,50.0,5.0,52.5
Adj. box coord.  = 0.0,49.21875,5.625,53.4375
Orig. box WKT    = POLYGON ((0.0 50.0, 5.0 50.0, 5.0 52.5, 0.0 52.5, 0.0 50.0))
Adj. box WKT     = POLYGON ((0.0 49.21875, 5.625 49.21875, 5.625 53.4375, 0.0 53.4375, 0.
↪0 49.21875))
Grid size  = 2160 x 1620 cells
with
  TILE      = 540
  LEVEL     = 7
  INV_RES   = 384
  RES (deg) = 0.0026041666666666665
  RES (m)   = 289.89450727414993
```

Note, to check bounding box WKTs, you can use the handy Wicket tool.

4.3 Cube computation

4.3.1 `xcube compute`

Synopsis

Compute a cube variable from other cube variables using a user-provided Python function.

```
$ xcube compute --help
```

```
Usage: xcube compute [OPTIONS] SCRIPT [CUBE]...

  Compute a cube variable from other cube variables in CUBEs using a user-
  provided Python function in SCRIPT.

  The SCRIPT must define a function named "compute":

      def compute(*input_vars: numpy.ndarray,
                  input_params: Mapping[str, Any] = None,
                  dim_coords: Mapping[str, np.ndarray] = None,
                  dim_ranges: Mapping[str, Tuple[int, int]] = None) \
                  -> numpy.ndarray:
          # Compute new numpy array from inputs
          # output_array = ...
          return output_array

  where input_vars are numpy arrays (chunks) in the order given by VARIABLES
  or given by the variable names returned by an optional "initialize"
  function that my be defined in SCRIPT too, see below. input_params is a
  mapping of parameter names to values according to PARAMS or the ones
  returned by the aforesaid "initialize" function. dim_coords is a mapping
  from dimension name to coordinate labels for the current chunk to be
  computed. dim_ranges is a mapping from dimension name to index ranges into
  coordinate arrays of the cube.

  The SCRIPT may define a function named "initialize":

      def initialize(input_cubes: Sequence[xr.Dataset],
                     input_var_names: Sequence[str],
                     input_params: Mapping[str, Any]) \
                     -> Tuple[Sequence[str], Mapping[str, Any]]:
          # Compute new variable names and/or new parameters
          # new_input_var_names = ...
          # new_input_params = ...
          return new_input_var_names, new_input_params

  where input_cubes are the respective CUBEs, input_var_names the respective
  VARIABLES, and input_params are the respective PARAMS. The "initialize"
  function can be used to validate the data cubes, extract the desired
  variables in desired order and to provide some extra processing parameters
  passed to the "compute" function.
```

(continues on next page)

```
    Note that if no input variable names are specified, no variables are
    passed to the "compute" function.

    The SCRIPT may also define a function named "finalize":

        def finalize(output_cube: xr.Dataset,
                     input_params: Mapping[str, Any]) \
                     -> Optional[xr.Dataset]:
            # Optionally modify output_cube and return it or return None
            return output_cube

    If defined, the "finalize" function will be called before the command
    writes the new cube and then exists. The functions may perform a cleaning
    up or perform side effects such as write the cube to some sink. If the
    functions returns None, the CLI will *not* write any cube data.

Options:
  --variables, --vars VARIABLES  Comma-separated list of variable names.
  -p, --params PARAMS            Parameters passed as 'input_params' dict to
                                 compute() and init() functions in SCRIPT.
  -o, --output OUTPUT            Output path. Defaults to 'out.zarr'
  -f, --format FORMAT            Output format.
  -N, --name NAME                Output variable's name.
  -D, --dtype DTYPE              Output variable's data type.
  --help
```

Example

```
$ xcube compute s3-olci-cube.zarr ./algoithms/s3-olci-ndvi.py
```

with ./algoithms/s3-olci-ndvi.py being:

```
# TODO
```

Python API

The related Python API function is xcube.core.compute.compute_cube().

4.4 Cube inspection

4.4.1 xcube dump

Synopsis

Dump contents of a dataset.

```
$ xcube dump --help
```

```
Usage: xcube dump [OPTIONS] INPUT

  Dump contents of an input dataset.

Options:
  --variable, --var VARIABLE
                              Name of a variable (multiple allowed).
  -E, --encoding              Dump also variable encoding information.
  --help                      Show this message and exit.
```

Example

```
$ xcube dump xcube_cube.zarr
```

4.4.2 xcube verify

Synopsis

Perform cube verification.

```
$ xcube verify --help
```

```
Usage: xcube verify [OPTIONS] CUBE

  Perform cube verification.

  The tool verifies that CUBE
  * defines the dimensions "time", "lat", "lon";
  * has corresponding "time", "lat", "lon" coordinate variables and that they
    are valid, e.g. 1-D, non-empty, using correct units;
  * has valid  bounds variables for "time", "lat", "lon" coordinate
    variables, if any;
  * has any data variables and that they are valid, e.g. min. 3-D, all have
    same dimensions, have at least dimensions "time", "lat", "lon".

  If INPUT is a valid xcube dataset, the tool returns exit code 0. Otherwise a
  violation report is written to stdout and the tool returns exit code 3.

Options:
  --help  Show this message and exit.
```

Python API

The related Python API functions are

- `xcube.core.verify.verify_cube()`, and

- `xcube.core.verify.assert_cube()`.

4.5 Cube data extraction

4.5.1 `xcube extract`

Synopsis

Extract cube points.

```
$ xcube extract --help
```

```
Usage: xcube extract [OPTIONS] CUBE POINTS

  Extract data points from an xcube dataset.

  Extracts data cells from CUBE at coordinates given in each POINTS record
  and writes the resulting values to given output path and format.

  POINTS must be a CSV file that provides at least the columns "lon", "lat",
  and "time". The "lon" and "lat" columns provide a point's location in
  decimal degrees. The "time" column provides a point's date or date-time.
  Its format should preferably be ISO, but other formats may work as well.

Options:
  -o, --output OUTPUT    Output path. If omitted, output is written to stdout.
  -f, --format FORMAT    Output format. Currently, only 'csv' is supported.
  -C, --coords           Include cube cell coordinates in output.
  -B, --bounds           Include cube cell coordinate boundaries (if any) in
                         output.
  -I, --indexes          Include cube cell indexes in output.
  -R, --refs             Include point values as reference in output.
  --help                 Show this message and exit.
```

Example

```
$ xcube extract xcube_cube.zarr -o point_data.csv -Cb --indexes --refs
```

Python API

Related Python API functions are

- `xcube.core.extract.get_cube_values_for_points()`,
- `xcube.core.extract.get_cube_point_indexes()`, and
- `xcube.core.extract.get_cube_values_for_indexes()`.

4.6 Cube manipulation

4.6.1 `xcube chunk`

Synopsis

(Re-)chunk xcube dataset.

```
$ xcube chunk --help
```

```
Usage: xcube chunk [OPTIONS] CUBE

  (Re-)chunk xcube dataset. Changes the external chunking of all variables
  of CUBE according to CHUNKS and writes the result to OUTPUT.

  Note: There is a possibly more efficient way to (re-)chunk datasets
  through the dedicated tool "rechunker", see
  https://rechunker.readthedocs.io.

Options:
  -o, --output OUTPUT  Output path. Defaults to 'out.zarr'
  -f, --format FORMAT  Format of the output. If not given, guessed from
                       OUTPUT.
  -p, --params PARAMS  Parameters specific for the output format. Comma-
                       separated list of <key>=<value> pairs.
  -C, --chunks CHUNKS  Chunk sizes for each dimension. Comma-separated list of
                       <dim>=<size> pairs, e.g. "time=1,lat=270,lon=270"
  --help               Show this message and exit.
```

Example

```
$ xcube chunk input_not_chunked.zarr -o output_rechunked.zarr --chunks "time=1,lat=270,
↪lon=270"
```

Python API

The related Python API function is *xcube.core.chunk.chunk_dataset()*.

4.6.2 `xcube edit`

Synopsis

Edit metadata of an xcube dataset.

```
$ xcube edit --help
```

```
Usage: xcube edit [OPTIONS] CUBE

  Edit the metadata of an xcube dataset. Edits the metadata of a given CUBE.
  The command currently works only for data cubes using ZARR format.

Options:
  -o, --output OUTPUT      Output path. The placeholder "{input}" will be
                           replaced by the input's filename without extension
                           (such as ".zarr"). Defaults to
                           "{input}-edited.zarr".
  -M, --metadata METADATA  The metadata of the cube is edited. The metadata to
                           be changed should be passed over in a single yml
                           file.
  -C, --coords             Update the metadata of the coordinates of the xcube
                           dataset.
  -I, --in-place           Edit the cube in place. Ignores output path.
  --help                   Show this message and exit.
```

Examples

The global attributes of the demo xcube dataset demo cube-1-250-250.zarr in the examples folder do not contain the creators name not an url. Furthermore the long name of the variable 'conc_chl' is 'Chlorophylll concentration', with too many l's. This can be fixed by using xcube edit. A yml-file defining the key words to be changed with the new content has to be created. The demo yml is saved in the examples folder.

Edit the metadata of the existing xcube dataset `cube-1-250-250-edited.zarr`:

```
$ xcube edit /examples/serve/demo/cube-1-250-250.zarr -M examples/edit/edit_metadata_
→cube-1-250-250.yml -o cube-1-250-250-edited.zarr
```

The global attributes below, which are related to the xcube dataset coodrinates cannot be manually edited.

- `geospatial_lon_min`
- `geospatial_lon_max`
- `geospatial_lon_units`
- `geospatial_lon_resolution`
- `geospatial_lat_min`
- `geospatial_lat_max`

- `geospatial_lat_units`

- `geospatial_lat_resolution`

- `time_coverage_start`

- `time_coverage_end`

If you wish to update these attributes, you can use the commandline parameter -C:

> $ xcube edit /examples/serve/demo/cube-1-250-250.zarr -C -o cube-1-250-250-edited.zarr

The -C will update the coordinate attributes based on information derived directly from the cube.

Python API

The related Python API function is *xcube.core.edit.edit_metadata()*.

4.6.3 `xcube level`

Synopsis

Generate multi-resolution levels.

```
$ xcube level --help
```

```
Usage: xcube level [OPTIONS] INPUT

  Generate multi-resolution levels.

  Transform the given dataset by INPUT into the levels of a multi-level
  pyramid with spatial resolution decreasing by a factor of two in both
  spatial dimensions and write the result to directory OUTPUT.

  INPUT may be an S3 object storage URL of the form "s3://<bucket>/<path>" or
  "https://<endpoint>".

Options:
  -o, --output OUTPUT             Output path. If omitted, "INPUT.levels" will
                                  be used. You can also use S3 object storage
                                  URLs of the form "s3://<bucket>/<path>" or
                                  "https://<endpoint>"
  -L, --link                      Link the INPUT instead of converting it to a
                                  level zero dataset. Use with care, as the
                                  INPUT's internal spatial chunk sizes may be
                                  inappropriate for imaging purposes.
  -t, --tile-size TILE_SIZE       Tile size, given as single integer number or
                                  as <tile-width>,<tile-height>. If omitted,
                                  the tile size will be derived from the
                                  INPUT's internal spatial chunk sizes. If the
                                  INPUT is not chunked, tile size will be 512.
  -n, --num-levels-max NUM_LEVELS_MAX
                                  Maximum number of levels to generate. If not
                                  given, the number of levels will be derived
```

	from spatial dimension and tile sizes.
-A, --agg-methods AGG_METHODS	Aggregation method(s) to be used for data variables. Either one of "first", "min", "max", "mean", "median", "auto" or list of assignments to individual variables using the notation "<var1>=<method1>,<var2>=<method2>,..." Defaults to "first".
-r, --replace	Whether to replace an existing dataset at OUTPUT.
-a, --anon	For S3 inputs or outputs, whether the access is anonymous. By default, credentials are required.
--help	Show this message and exit.

Example

```
$ xcube level --link -t 720 data/cubes/test-cube.zarr
```

Python API

The related Python API functions are

- `xcube.core.level.compute_levels()`,
- `xcube.core.level.read_levels()`, and
- `xcube.core.level.write_levels()`.

4.6.4 xcube optimize

Synopsis

Optimize xcube dataset for faster access.

```
$ xcube optimize --help
```

```
Usage: xcube optimize [OPTIONS] CUBE

  Optimize xcube dataset for faster access.

  Reduces the number of metadata and coordinate data files in xcube dataset
  given by CUBE. Consolidated cubes open much faster especially from remote
  locations, e.g. in object storage, because obviously much less HTTP
  requests are required to fetch initial cube meta information. That is, it
  merges all metadata files into a single top-level JSON file ".zmetadata".
  Optionally, it removes any chunking of coordinate variables so they
  comprise a single binary data file instead of one file per data chunk. The
  primary usage of this command is to optimize data cubes for cloud object
  storage. The command currently works only for data cubes using ZARR
```

```
    format.

Options:
  -o, --output OUTPUT  Output path. The placeholder "<built-in function
                       input>" will be replaced by the input's filename
                       without extension (such as ".zarr"). Defaults to
                       "{input}-optimized.zarr".
  -I, --in-place       Optimize cube in place. Ignores output path.
  -C, --coords         Also optimize coordinate variables by converting any
                       chunked arrays into single, non-chunked, contiguous
                       arrays.
  --help               Show this message and exit.
```

Examples

Write an cube with consolidated metadata to `cube-optimized.zarr`:

```
$ xcube optimize ./cube.zarr
```

Write an optimized cube with consolidated metadata and consolidated coordinate variables to `optimized/cube.zarr`
(directory `optimized` must exist):

```
$ xcube optimize -C -o ./optimized/cube.zarr ./cube.zarr
```

Optimize a cube in-place with consolidated metadata and consolidated coordinate variables:

```
$ xcube optimize -IC ./cube.zarr
```

Python API

The related Python API function is *xcube.core.optimize.optimize_dataset()*.

4.6.5 `xcube patch`

Synopsis

Patch and consolidate the metadata of a xcube dataset.

```
$ xcube patch --help
```

```
Usage: xcube patch [OPTIONS] DATASET

  Patch and consolidate the metadata of a dataset.

  DATASET can be either a local filesystem path or a URL. It must point to
  either a Zarr dataset (*.zarr) or a xcube multi-level dataset (*.levels).
  Additional storage options for a given protocol may be passed by the OPTIONS
  option.
```

```
  In METADATA, the special attribute value "__delete__" can be used to remove
  that attribute from dataset or array metadata.

Options:
  --metadata METADATA   The metadata to be patched. Must be a JSON or YAML file
                        using Zarr consolidated metadata format.
  --options OPTIONS     Protocol-specific storage options (see fsspec). Must be
                        a JSON or YAML file.
  -q, --quiet           Disable output of log messages to the console entirely.
                        Note, this will also suppress error and warning
                        messages.
  -v, --verbose         Enable output of log messages to the console. Has no
                        effect if --quiet/-q is used. May be given multiple
                        times to control the level of log messages, i.e., -v
                        refers to level INFO, -vv to DETAIL, -vvv to DEBUG,
                        -vvvv to TRACE. If omitted, the log level of the
                        console is WARNING.
  -d, --dry-run         Do not change any data, just report what would have
                        been changed.
  --help                Show this message and exit.
```

Patch file example

Patch files use the Zarr Consolidated Metadata Format, v1. For example, the following patch file (YAML) will delete the global attribute `TileSize` and change the value of the attribute `long_name` of variable `conc_chl`:

```yaml
zarr_consolidated_format: 1
metadata:

  .zattrs:
    TileSize: __delete__

  conc_chl/.zattrs:
    long_name: Chlorophyll concentration
```

Storage options file example

Here is a storage options file for the "s3" protocol that provides credentials for AWS S3 access:

```yaml
key: AJDKJCLSKKA
secret: kjkl456lkj45632k45j63l
```

Usage example

```
$ xcube patch s3://my-cubes-bucket/test.zarr --metadata patch.yml -v
```

4.6.6 `xcube prune`

Delete empty chunks.

> **Attention:** This tool will likely be integrated into `xcube optimize` in the near future.

```
$ xcube prune --help
```

```
Usage: xcube prune [OPTIONS] DATASET

  Delete empty chunks. Deletes all data files associated with empty (NaN-
  only) chunks in given DATASET, which must have ZARR format.

Options:
  -v, --verbose  Verbose mode. Multiple may be given, for example "-vvv".
  --dry-run      Just read and process input, but don't produce any outputs.
  --help         Show this message and exit.
```

A related Python API function is `xcube.core.optimize.get_empty_dataset_chunks()`.

4.6.7 `xcube resample`

Synopsis

Resample data along the time dimension.

```
$ xcube resample --help
```

```
Usage: xcube resample [OPTIONS] CUBE

  Resample data along the time dimension.

Options:
  -c, --config CONFIG             xcube dataset configuration file in YAML
                                  format. More than one config input file is
                                  allowed.When passing several config files,
                                  they are merged considering the order passed
                                  via command line.
  -o, --output OUTPUT             Output path. Defaults to 'out.zarr'.
  -f, --format [zarr|netcdf4|mem]
                                  Output format. If omitted, format will be
                                  guessed from output path.
  --variables, --vars VARIABLES   Comma-separated list of names of variables
                                  to be included.
```

`-M, --method TEXT`	Temporal resampling method. Available downsampling methods are `'count'`, `'first'`, `'last'`, `'min'`, `'max'`, `'sum'`, `'prod'`, `'mean'`, `'median'`, `'std'`, `'var'`, the upsampling methods are `'asfreq'`, `'ffill'`, `'bfill'`, `'pad'`, `'nearest'`, `'interpolate'`. If the upsampling method **is** `'interpolate'`, the option `'--kind'` will be used, **if** given. Other upsampling methods that select existing values honour the `'--tolerance'` option. Defaults to `'mean'`.
`-F, --frequency TEXT`	Temporal aggregation frequency. Use format `"<count><offset>"` where `<offset>` **is** one of `'H'`, `'D'`, `'W'`, `'M'`, `'Q'`, `'Y'`. Defaults to `'1D'`.
`-O, --offset TEXT`	Offset used to adjust the resampled time labels. Uses same syntax **as** frequency. Some Pandas date offset strings are supported **as** well.
`-B, --base INTEGER`	For frequencies that evenly subdivide `1` day, the origin of the aggregated intervals. For example, **for** `'24H'` frequency, base could range **from** `0` through `23`. Defaults to `0`.
`-K, --kind TEXT`	Interpolation kind which will be used **if** upsampling method **is** `'interpolation'`. May be one of `'zero'`, `'slinear'`, `'quadratic'`, `'cubic'`, `'linear'`, `'nearest'`, `'previous'`, `'next'` where `'zero'`, `'slinear'`, `'quadratic'`, `'cubic'` refer to a spline interpolation of zeroth, first, second **or** third order; `'previous'` **and** `'next'` simply **return** the previous **or** next value of the point. For more info refer to scipy.interpolate.interp1d(). Defaults to `'linear'`.
`-T, --tolerance TEXT`	Tolerance **for** selective upsampling methods. Uses same syntax **as** frequency. If the time delta exceeds the tolerance, fill values (NaN) will be used. Defaults to the given frequency.
`--dry-run`	Just read **and** process inputs, but don`'`t produce **any** outputs.
`--help`	Show this message **and** exit.

Examples

Upsampling example:

```
$ xcube resample --vars conc_chl,conc_tsm -F 12H -T 6H -M interpolation -K linear␣
→examples/serve/demo/cube.nc
```

Downsampling example:

```
$ xcube resample --vars conc_chl,conc_tsm -F 3D -M mean -M std -M count examples/serve/
→demo/cube.nc
```

Python API

The related Python API function is xcube.core.resample.resample_in_time().

4.6.8 xcube vars2dim

Synopsis

Convert cube variables into new dimension.

```
$ xcube vars2dim --help
```

```
Usage: xcube vars2dim [OPTIONS] CUBE

  Convert cube variables into new dimension. Moves all variables of CUBE
  into into a single new variable <var-name> with a new dimension DIM-NAME
  and writes the results to OUTPUT.

Options:
  --variable, --var VARIABLE  Name of the new variable that includes all
                              variables. Defaults to "data".
  -D, --dim_name DIM-NAME     Name of the new dimension into variables.
                              Defaults to "var".
  -o, --output OUTPUT         Output path. If omitted, 'INPUT-vars2dim.INPUT-
                              FORMAT' will be used.
  -f, --format FORMAT         Format of the output. If not given, guessed from
                              OUTPUT.
  --help                      Show this message and exit.
```

Python API

The related Python API function is xcube.core.vars2dim.vars_to_dim().

4.7 Cube conversion

4.7.1 `xcube tile`

IMPORTANT NOTE: The xcube tile tool in its current form is deprecated and no longer supported since xcube 0.11. A new tool is planned that can work concurrently on dask clusters and also supports common tile grids such as global geographic and web mercator.

Synopsis

Generate a tiled RGB image pyramid from any xcube dataset.

The format and file organisation of the generated tile sets conforms to the TMS 1.0 Specification.

An optional configuration file given by the *-c* option uses YAML format.

```
$ xcube tile --help
```

```
Usage: xcube tile [OPTIONS] CUBE

  Create RGBA tiles from CUBE.

  Color bars and value ranges for variables can be specified in a CONFIG file.
  Here the color mappings are defined for a style named "ocean_color":

  Styles:
    - Identifier: ocean_color
      ColorMappings:
        conc_chl:
          ColorBar: "plasma"
          ValueRange: [0., 24.]
        conc_tsm:
          ColorBar: "PuBuGn"
          ValueRange: [0., 100.]
        kd489:
          ColorBar: "jet"
          ValueRange: [0., 6.]

  This is the same styles syntax as the configuration file for "xcube serve",
  hence its configuration can be reused.

Options:
  --variables, --vars VARIABLES  Variables to be included in output. Comma-
                                 separated list of names which may contain
                                 wildcard characters "*" and "?".
  --labels LABELS                Labels for non-spatial dimensions, e.g.
                                 "time=2019-20-03". Multiple values are
                                 separated by comma.
  -t, --tile-size TILE_SIZE      Tile size in pixels for individual or both x-
                                 and y-directions. Separate by comma for
                                 individual tile sizes, e.g. "-t 360,180".
                                 Defaults to the chunks sizes in x- and
```

(continues on next page)

```
                                 y-directions of CUBE, which may not be ideal.
                                 Use option --dry-run and --verbose to display
                                 the default tile sizes for CUBE.
 -c, --config CONFIG             Configuration file in YAML format.
 -s, --style STYLE               Name of a style identifier in CONFIG file.
                                 Only used if CONFIG is given. Defaults to
                                 'default'.
 -o, --output OUTPUT             Output path. Defaults to 'out.tiles'
 -v, --verbose                   Use -vv to report all files generated, -v to
                                 report less.
 --dry-run                       Generate all tiles but don't write any files.
 --help                          Show this message and exit.
```

Example

An example that uses a configuration file only:

````
```bash
````

> xcube tile https://s3.eu-central-1.amazonaws.com/esdl-esdc-v2.0.0/esdc-8d-0.083deg-1x2160x4320-2.0.0.zarr
> –labels time='2009-01-01/2009-12-30' –vars analysed_sst,air_temperature_2m –tile-size 270 –con-
> fig ./config-cci-cfs.yml –style default –verbose

` ` `

The configuration file *config-cci-cfs.yml* content is:

` ` `**yaml**

> **Styles:**
>
>> • Identifier: default ColorMappings:
>>
>>> **analysed_sst:** ColorBar: "inferno" ValueRange: [270, 310]
>>>
>>> **air_temperature_2m:** ColorBar: "magma" ValueRange: [190, 320]

` ` `

Python API

There is currently no related Python API.

4.8 Cube publication

4.8.1 `xcube serve`

Synopsis

Serve data cubes via web service.

`xcube serve` starts a light-weight web server that provides various services based on xcube datasets:

 • Catalogue services to query for xcube datasets and their variables and dimensions, and feature collections;

- Tile map service, with some OGC WMTS 1.0 compatibility (REST and KVP APIs);

- Dataset services to extract subsets like time-series and profiles for e.g. JavaScript clients.

```
$ xcube serve --help
```

```
Usage: xcube serve [OPTIONS] [CUBE]...

  Serve data cubes via web service.

  Serves data cubes by a RESTful API and a OGC WMTS 1.0 RESTful and KVP
  interface. The RESTful API documentation can be found at
  https://app.swaggerhub.com/apis/bcdev/xcube-server.

Options:
  -A, --address ADDRESS    Service address. Defaults to 'localhost'.
  -P, --port PORT          Port number where the service will listen on.
                           Defaults to 8080.
  --prefix PREFIX          Service URL prefix. May contain template patterns
                           such as "${version}" or "${name}". For example
                           "${name}/api/${version}". Will be used to prefix
                           all API operation routes and in any URLs returned
                           by the service.
  --revprefix REVPREFIX    Service reverse URL prefix. May contain template
                           patterns such as "${version}" or "${name}". For
                           example "${name}/api/${version}". Defaults to
                           PREFIX, if any. Will be used only in URLs returned
                           by the service e.g. the tile URLs returned by the
                           WMTS service.
  -u, --update PERIOD      Service will update after given seconds of
                           inactivity. Zero or a negative value will disable
                           update checks. Defaults to 2.0.
  -S, --styles STYLES      Color mapping styles for variables. Used only, if
                           one or more CUBE arguments are provided and CONFIG
                           is not given. Comma-separated list with elements of
                           the form <var>=(<vmin>,<vmax>) or
                           <var>=(<vmin>,<vmax>,"<cmap>")
  -c, --config CONFIG      Use datasets configuration file CONFIG. Cannot be
                           used if CUBES are provided. If not given and also
                           CUBES are not provided, the configuration may be
                           given by environment variable
                           XCUBE_SERVE_CONFIG_FILE.
  -b, --base-dir BASE_DIR  Base directory used to resolve relative dataset
                           paths in CONFIG and relative CUBES paths. Defaults
                           to value of environment variable
                           XCUBE_SERVE_BASE_DIR, if given, otherwise defaults
                           to the parent directory of CONFIG.
  --tilecache SIZE         In-memory tile cache size in bytes. Unit suffixes
                           'K', 'M', 'G' may be used. Defaults to '512M'. The
                           special value 'OFF' disables tile caching.
  --tilemode MODE          Tile computation mode. This is an internal option
                           used to switch between different tile computation
                           implementations. Defaults to 0.
  -s, --show               Run viewer app. Requires setting the environment
```

(continues on next page)

```
                          variable XCUBE_VIEWER_PATH to a valid xcube-viewer
                          deployment or build directory. Refer to
                          https://github.com/dcs4cop/xcube-viewer for more
                          information.
 -q, --quiet              Disable output of log messages to the console
                          entirely. Note, this will also suppress error and
                          warning messages.
 -v, --verbose            Enable output of log messages to the console. Has
                          no effect if --quiet/-q is used. May be given
                          multiple times to control the level of log
                          messages, i.e., -v refers to level INFO, -vv to
                          DETAIL, -vvv to DEBUG, -vvvv to TRACE. If omitted,
                          the log level of the console is WARNING.
 --traceperf              Log extra performance diagnostics using log level
                          DEBUG.
 --aws-prof PROFILE       To publish remote CUBEs, use AWS credentials from
                          section [PROFILE] found in ~/.aws/credentials.
 --aws-env                To publish remote CUBEs, use AWS credentials from
                          environment variables AWS_ACCESS_KEY_ID and
                          AWS_SECRET_ACCESS_KEY
 --help                   Show this message and exit.
```

Configuration File

The xcube server is used to configure the xcube datasets to be published.

xcube datasets are any datasets that

- that comply to Unidata's CDM and to the CF Conventions;

- that can be opened with the xarray Python library;

- that have variables that have the dimensions and shape (lat, lon) or (time, lat, lon);

- that have 1D-coordinate variables corresponding to the dimensions;

- that have their spatial grid defined in arbitrary spatial coordinate reference systems.

The xcube server supports xcube datasets stored as local NetCDF files, as well as Zarr directories in the local file system or remote object storage. Remote Zarr datasets must be stored AWS S3 compatible object storage.

As an example, here is the configuration of the demo server. The parts of the demo configuration file are explained in detail further down.

Some hints before, which are not addressed in the server demo configuration file. To increase imaging performance, xcube datasets can be converted to multi-resolution pyramids using the *xcube level* tool. In the configuration, the format must be set to 'level'. Leveled xcube datasets are configured this way:

```
Datasets:

  - Identifier: my_multi_level_dataset
    Title: "My Multi-Level Dataset"
    FileSystem: file
    Path: my_multi_level_dataset.level

  - ...
```

To increase time-series extraction performance, xcube datasets may be rechunked with larger chunk size in the `time` dimension using the *xcube chunk* tool. In the xcube server configuration a hidden dataset is given, and the it is referred to by the non-hidden, actual dataset using the `TimeSeriesDataset` setting:

```
Datasets:

  - Identifier: my_dataset
    Title: "My Dataset"
    FileSystem: file
    Path: my_dataset.zarr
    TimeSeriesDataset: my_dataset_opt_for_ts

  - Identifier: my_dataset_opt_for_ts
    Title: "My Dataset optimized for Time-Series"
    FileSystem: file
    Path: my_ts_opt_dataset.zarr
    Hidden: True

  - ...
```

Server Demo Configuration File

The server configuration file consists of various parts, some of them are necessary others are optional. Here the demo configuration file used in the *example* is explained in detail.

The configuration file consists of five main parts *authentication*, *dataset attribution*, *datasets*, *place groups* and *styles*.

Authentication [optional]

In order to display data via xcube-viewer exclusively to registered and authorized users, the data served by xcube serve may be protected by adding Authentication to the server configuration. In order to ensure protection, an *Authority* and an *Audience* needs to be provided. Here authentication by Auth0 is used. Please note the trailing slash in the "Authority" URL.

```
Authentication:
  Authority: https://xcube-dev.eu.auth0.com/
  Audience: https://xcube-dev/api/
```

Example of OIDC configuration for Keycloak. Please note that there is no trailing slash in the "Authority" URL.

```
Authentication:
  Authority: https://kc.brockmann-consult.de/auth/realms/AVL
  Audience: avl-xc-api
```

Dataset Attribution [optional]

Dataset Attribution may be added to the server via *DatasetAttribution*.

```
DatasetAttribution:
  - "© by Brockmann Consult GmbH 2020, contains modified Copernicus Data 2019, processed␣
→by ESA"
  - "© by EU H2020 CyanoAlert project"
```

Datasets [mandatory]

In order to publish selected xcube datasets via xcube serve the datasets need to be specified in the configuration file of the server. Several xcube datasets may be served within one server, by providing a list of information concerning the xcube datasets.

Remotely Stored xcube Datasets

```
Datasets:
  - Identifier: remote
    Title: Remote OLCI L2C cube for region SNS
    BoundingBox: [0.0, 50, 5.0, 52.5]
    FileSystem: s3
    Endpoint: "https://s3.eu-central-1.amazonaws.com"
    Path: "xcube-examples/OLCI-SNS-RAW-CUBE-2.zarr"
    Region: "eu-central-1"
    Anonymous: true
    Style: default
    ChunkCacheSize: 250M
    PlaceGroups:
      - PlaceGroupRef: inside-cube
      - PlaceGroupRef: outside-cube
    AccessControl:
      RequiredScopes:
        - read:datasets
```

The above example of how to specify a xcube dataset to be served above is using a datacube stored in an S3 bucket within the Amazon Cloud. Please have a closer look at the parameter *Anonymous: true*. This means, the datasets permissions are set to public read in your source s3 bucket. If you have a dataset that is not public-read, set *Anonymous: false*. Furthermore, you need to have valid credentials on the machine where the server runs. Credentials may be saved either in a file called .aws/credentials with content like below:

> [default]
>
> aws_access_key_id=AKIAIOSFODNN7EXAMPLE
>
> aws_secret_access_key=wJalrXUtnFEMI/K7MDENG/bPxRfiCYEXAMPLEKEY

Or they may be exported as environment variables AWS_SECRET_ACCESS_KEY and AWS_ACCESS_KEY_ID.

Further down an example for a *locally stored xcube datasets* will be given, as well as an example of a *on-the-fly generation of xcube datasets*.

Identifier [mandatory] is a unique ID for each xcube dataset, it is ment for machine-to-machine interaction and therefore does not have to be a fancy human-readable name.

Title [optional] should be understandable for humans. This title that will be displayed within the viewer for the dataset selection. If omitted, the key title from the dataset metadata will be used. If that is missing too, the identifier will be used.

BoundingBox [optional] may be set in order to restrict the region which is served from a certain datacube. The notation of the *BoundingBox* is [lon_min,lat_min,lon_max,lat_max].

FileSystem [mandatory] is set to "s3" which lets xcube serve know, that the datacube is located in the cloud.

Endpoint [mandatory] contains information about the cloud provider endpoint, this will differ if you use a different cloud provider.

Path [mandatory] leads to the specific location of the datacube. The particular datacube is stored in an OpenTelecom-Cloud S3 bucket called "xcube-examples" and the datacube is called "OLCI-SNS-RAW-CUBE-2.zarr".

Region [optional] is the region where the specified cloud provider is operating.

Styles [optional] influence the visualization of the xucbe dataset in the xcube viewer if specified in the server configuration file. The usage of *Styles* is described in section *styles*.

PlaceGroups [optional] allow to associate places (e.g. polygons or point-location) with a particular xcube dataset. Several different place groups may be connected to a xcube dataset, these different place groups are distinguished by the *PlaceGroupRef*. The configuration of *PlaceGroups* is described in section *place groups*.

AccessControl [optional] can only be used when providing *authentication*. Datasets may be protected by configuring the *RequiredScopes* entry whose value is a list of required scopes, e.g. "read:datasets".

Locally Stored xcube Datasets

To serve a locally stored dataset the configuration of it would look like the example below:

```
- Identifier: local
  Title: "Local OLCI L2C cube for region SNS"
  BoundingBox: [0.0, 50, 5.0, 52.5]
  FileSystem: file
  Path: cube-1-250-250.zarr
  Style: default
  TimeSeriesDataset: local_ts
  Augmentation:
    Path: "compute_extra_vars.py"
    Function: "compute_variables"
    InputParameters:
      factor_chl: 0.2
      factor_tsm: 0.7
  PlaceGroups:
    - PlaceGroupRef: inside-cube
    - PlaceGroupRef: outside-cube
  AccessControl:
    IsSubstitute: true
```

Most of the configuration of locally stored datasets is equal to the configuration of *remotely stored xcube datasets*.

FileSystem [mandatory] is set to "file" which lets xcube serve know, that the datacube is locally stored.

TimeSeriesDataset [optional] is not bound to local datasets, this parameter may be used for remotely stored datasets as well. By using this parameter a time optimized datacube will be used for generating the time series. The configuration of this time optimized datacube is shown below. By adding *Hidden* with *true* to the dataset configuration, the time optimized datacube will not appear among the displayed datasets in xcube viewer.

```
# Will not appear at all, because it is a "hidden" resource
- Identifier: local_ts
  Title: "'local' optimized for time-series"
  BoundingBox: [0.0, 50, 5.0, 52.5]
  FileSystem: file
  Path: cube-5-100-200.zarr
  Hidden: true
  Style: default
```

Augmentation [optional] augments data cubes by new variables computed on-the-fly, the generation of the on-the-fly variables depends on the implementation of the python module specified in the *Path* within the *Augmentation* configuration.

AccessControl [optional] can only be used when providing *authentication*. By passing the *IsSubstitute* flag a dataset disappears for authorized requests. This might be useful for showing a demo dataset in the viewer for user who are not logged in.

On-the-fly Generation of xcube Datasets

There is the possibility of generating resampled xcube datasets on-the-fly, e.g. in order to obtain daily or weekly averages of a xcube dataset.

```
- Identifier: local_1w
  Title: OLCI weekly L3 cube for region SNS computed from local L2C cube
  BoundingBox: [0.0, 50, 5.0, 52.5]
  FileSystem: memory
  Path: "resample_in_time.py"
  Function: "compute_dataset"
  InputDatasets: ["local"]
  InputParameters:
    period: "1W"
    incl_stdev: True
  Style: default
  PlaceGroups:
    - PlaceGroupRef: inside-cube
    - PlaceGroupRef: outside-cube
  AccessControl:
    IsSubstitute: True
```

FileSystem [mandatory] is defined as "memory" for the on-the-fly generated dataset.

Path [mandatory] leads to the resample python module. There might be several functions specified in the python module, therefore the particular *Function* needs to be included into the configuration.

InputDatasets [mandatory] specifies the dataset to be resampled.

InputParameter [mandatory] defines which kind of resampling should be performed. In the example a weekly average is computed.

Again, the dataset may be associated with place groups.

Place Groups [optional]

Place groups are specified in a similar manner compared to specifying datasets within a server. Place groups may be stored e.g. in shapefiles or a geoJson.

```
PlaceGroups:
  - Identifier: outside-cube
    Title: Points outside the cube
    Path: "places/outside-cube.geojson"
    PropertyMapping:
      image: "${base_url}/images/outside-cube/${ID}.jpg"
```

Identifier [mandatory] is a unique ID for each place group, it is the one xcube serve uses to associate a place group to a particular dataset.

Title [mandatory] should be understandable for humans and this is the title that will be displayed within the viewer for the place selection if the selected xcube dataset contains a place group.

Path [mandatory] defines where the file storing the place group is located. Please note that the paths within the example config are relative.

PropertyMapping [mandatory] determines which information contained within the place group should be used for selecting a certain location of the given place group. This depends very strongly of the data used. In the above example, the image URL is determined by a feature's ID property.

Property Mappings

The entry *PropertyMapping* is used to map a set of well-known properties (or roles) to the actual properties provided by a place feature in a place group. For example, the well-known properties are used to in xcube viewer to display information about the currently selected place. The possible well-known properties are:

- `label`: The property that provides a label for the place, if any. Defaults to to case-insensitive names `label`, `title`, `name`, `id` in xcube viewer.

- `color`: The property that provides a place's color. Defaults to the case-insensitive name `color` in xcube viewer.

- `image`: The property that provides a place's image URL, if any. Defaults to case-insensitive names `image`, `img`, `picture`, `pic` in xcube viewer.

- `description`: The property that provides a place's description text, if any. Defaults to case-insensitive names `description`, `desc`, `abstract`, `comment` in xcube viewer.

In the following example, a place's label is provided by the place feature's `NAME` property, while an image is provided by the place feature's `IMG_URL` property:

```
PlaceGroups:
    Identifier: my_group
    ...
    PropertyMapping:
        label: NAME
        image: IMG_URL
```

The values on the right side may either **be** feature property names or **contain** them as placeholders in the form `${PROPERTY}`. A special placeholder is `${base_url}` which is replaced by the server's current base URL.

Styles [optional]

Within the *Styles* section colorbars may be defined which should be used initially for a certain variable of a dataset, as well as the value ranges. For xcube viewer version 0.3.0 or higher the colorbars and the value ranges may be adjusted by the user within the xcube viewer.

```
Styles:
  - Identifier: default
    ColorMappings:
      conc_chl:
        ColorBar: "plasma"
        ValueRange: [0., 24.]
      conc_tsm:
        ColorBar: "PuBuGn"
        ValueRange: [0., 100.]
      kd489:
        ColorBar: "jet"
        ValueRange: [0., 6.]
      rgb:
        Red:
          Variable: conc_chl
          ValueRange: [0., 24.]
        Green:
          Variable: conc_tsm
          ValueRange: [0., 100.]
        Blue:
          Variable: kd489
          ValueRange: [0., 6.]
```

The *ColorMapping* may be specified for each variable of the datasets to be served. If not specified, the server uses a default colorbar as well as a default value range.

rgb may be used to generate an RGB-Image on-the-fly within xcube viewer. This may be done if the dataset contains variables which represent the bands red, green and blue, they may be combined to an RGB-Image. Or three variables of the dataset may be combined to an RGB-Image, as shown in the configuration above.

Example

```
xcube serve --port 8080 --config ./examples/serve/demo/config.yml --verbose
```

```
xcube Server: WMTS, catalogue, data access, tile, feature, time-series services for
→xarray-enabled data cubes, version 0.2.0
[I 190924 17:08:54 service:228] configuration file 'D:\\Projects\\xcube\\examples\\serve\
→\demo\\config.yml' successfully loaded
[I 190924 17:08:54 service:158] service running, listening on localhost:8080, try http://
→localhost:8080/datasets
[I 190924 17:08:54 service:159] press CTRL+C to stop service
```

Server Demo Configuration File for DataStores

The server configuration file consists of various parts, some of them are necessary, others are optional. Here the demo stores configuration file used in the *example stores* is explained in detail.

This configuration file differs only in one part compared to section *Server Demo Configuration File*: *data stores*. The other main parts (*authentication*, *dataset attribution*, *place groups*, and *styles*) can be used in combination with *data stores*.

DataStores [mandatory]

Datasets, which are stored in the same location, may be configured in the configuration file using *DataStores*.

```
DataStores:
  - Identifier: edc
    StoreId: s3
    StoreParams:
      root: xcube-dcfs/edc-xc-viewer-data
      max_depth: 1
      storage_options:
        anon: true
        # client_kwargs:
        #   endpoint_url: https://s3.eu-central-1.amazonaws.com
    Datasets:
      - Path: "*2.zarr"
        Style: "default"
        # ChunkCacheSize: 1G
```

Identifier [mandatory] is a unique ID for each DataStore.

StoreID [mandatory] can be *file* for locally stored datasets and *s3* for datasets located in the cloud.

StoreParams [mandatory]

root [mandatory] states a common beginning part of the paths of the served datasets.

max_depth [optional] if wildcard is used in *Dataset Path* this indicated how far the server should step down and serve the discovered datasets.

storage_options [optional] is necessary when serving datasets from the cloud. With *anon* the accessibility is configured, if the datasets are public-read, *anon* is set to "true", "false" indicates they are protected. Credentials may be set by keywords *key* and *secret*.

Datasets [optional] if not specified, every dataset in the indicated location supported by xcube will be read and served by xcube serve. In order to filter certain datasets you can list Paths that shall be served by xcube serve. *Path* may contain wildcards. Each Dataset entry may have *Styles* and *PlaceGroups* associated with them, the same way as described in section *Server Demo Configuration File*.

Example Stores

```
xcube serve --port 8080 --config ./examples/serve/demo/config-with-stores.yml --verbose
```

```
xcube Server: WMTS, catalogue, data access, tile, feature, time-series services for␣
↪xarray-enabled data cubes, version
[I 190924 17:08:54 service:228] configuration file 'D:\\Projects\\xcube\\examples\\serve\
↪\demo\\config.yml' successfully loaded
[I 190924 17:08:54 service:158] service running, listening on localhost:8080, try http://
↪localhost:8080/datasets
[I 190924 17:08:54 service:159] press CTRL+C to stop service
```

Web API

The xcube server has a dedicated Web API Documentation on SwaggerHub. It also allows you to explore the API of existing xcube-servers.

The xcube server implements the OGC WMTS RESTful and KVP architectural styles of the OGC WMTS 1.0.0 specification. The following operations are supported:

- **GetCapabilities**: /xcube/wmts/1.0.0/WMTSCapabilities.xml

- **GetTile**: /xcube/wmts/1.0.0/tile/{DatasetName}/{VarName}/{TileMatrix}/{TileCol}/{TileRow}.png

- **GetFeatureInfo**: *in progress*

PYTHON API

5.1 Cube I/O

5.2 Cube generation

xcube.core.new.**new_cube**(*title='Test Cube'*, *width=360*, *height=180*, *x_name='lon'*, *y_name='lat'*, *x_dtype='float64'*, *y_dtype=None*, *x_units='degrees_east'*, *y_units='degrees_north'*, *x_res=1.0*, *y_res=None*, *x_start=- 180.0*, *y_start=- 90.0*, *inverse_y=False*, *time_name='time'*, *time_dtype='datetime64[s]'*, *time_units='seconds since 1970-01-01T00:00:00'*, *time_calendar='proleptic_gregorian'*, *time_periods=5*, *time_freq='D'*, *time_start='2010-01-01T00:00:00'*, *use_cftime=False*, *drop_bounds=False*, *variables=None*, *crs=None*, *crs_name=None*)

Create a new empty cube. Useful for creating cubes templates with predefined coordinate variables and metadata. The function is also heavily used by xcube's unit tests.

The values of the *variables* dictionary can be either constants, array-like objects, or functions that compute their return value from passed coordinate indexes. The expected signature is::

```python
def my_func(time: int, y: int, x: int) -> Union[bool, int, float]
```

> **Parameters**
>
> - **title** (str) – A title. Defaults to 'Test Cube'.
> - **width** (int) – Horizontal number of grid cells. Defaults to 360.
> - **height** (int) – Vertical number of grid cells. Defaults to 180.
> - **x_name** (str) – Name of the x coordinate variable. Defaults to 'lon'.
> - **y_name** (str) – Name of the y coordinate variable. Defaults to 'lat'.
> - **x_dtype** (str) – Data type of x coordinates. Defaults to 'float64'.
> - **y_dtype** – Data type of y coordinates. Defaults to 'float64'.
> - **x_units** (str) – Units of the x coordinates. Defaults to 'degrees_east'.
> - **y_units** (str) – Units of the y coordinates. Defaults to 'degrees_north'.
> - **x_start** (float) – Minimum x value. Defaults to -180.
> - **y_start** (float) – Minimum y value. Defaults to -90.
> - **x_res** (float) – Spatial resolution in x-direction. Defaults to 1.0.
> - **y_res** – Spatial resolution in y-direction. Defaults to 1.0.

- **inverse_y** (`bool`) – Whether to create an inverse y axis. Defaults to False.

- **time_name** (`str`) – Name of the time coordinate variable. Defaults to 'time'.

- **time_periods** (`int`) – Number of time steps. Defaults to 5.

- **time_freq** (`str`) – Duration of each time step. Defaults to `` `1D' ``.

- **time_start** (`str`) – First time value. Defaults to '2010-01-01T00:00:00'.

- **time_dtype** (`str`) – Numpy data type for time coordinates. Defaults to 'datetime64[s]'. If used, parameter 'use_cftime' must be False.

- **time_units** (`str`) – Units for time coordinates. Defaults to 'seconds since 1970-01-01T00:00:00'.

- **time_calendar** (`str`) – Calender for time coordinates. Defaults to 'proleptic_gregorian'.

- **use_cftime** (`bool`) – If True, the time will be given as data types according to the 'cftime' package. If used, the time_calendar parameter must be also be given with an appropriate value such as 'gregorian' or 'julian'. If used, parameter 'time_dtype' must be None.

- **drop_bounds** (`bool`) – If True, coordinate bounds variables are not created. Defaults to False.

- **variables** – Dictionary of data variables to be added. None by default.

- **crs** – pyproj-compatible CRS string or instance of pyproj.CRS or None

- **crs_name** – Name of the variable that will hold the CRS information. Ignored, if *crs* is not given.

> **Returns** A cube instance

5.3 Cube computation

xcube.core.evaluate.**evaluate_dataset**(*dataset: [xarray.Dataset](), processed_variables: Optional[List[Tuple[[str](), Optional[Dict[[str](), Any]]]]] = None, errors: [str]() = 'raise'*) → [xarray.Dataset]()

Compute new variables or mask existing variables in *dataset* by the evaluation of Python expressions, that may refer to other existing or new variables. Returns a new dataset that contains the old and new variables, where both may bew now masked.

Expressions may be given by attributes of existing variables in *dataset* or passed a via the *processed_variables* argument which is a sequence of variable name / attributes tuples.

Two types of expression attributes are recognized in the attributes:

1. The attribute `expression` generates a new variable computed from its attribute value.

2. The attribute `valid_pixel_expression` masks out invalid variable values.

In both cases the attribuite value must be a string that forms a valid Python expression that can reference any other preceding variables by name. The expression can also reference any flags defined by another variable according the their CF attributes `flag_meaning` and `flag_values`.

Invalid variable values may be masked out using the value the `valid_pixel_expression` attribute whose value should form a Boolean Python expression. In case, the expression returns zero or false, the value of the `_FillValue` attribute or NaN will be used in the new variable.

Other attributes will be stored as variable metadata as-is.

> **Parameters**

- **dataset** – A dataset.

- **processed_variables** – Optional list of variable name-attributes pairs that will processed in the given order.

- **errors** (`str`) – How to deal with errors while evaluating expressions. May be be one of "raise", "warn", or "ignore".

Returns new dataset with computed variables

5.4 Cube data extraction

5.5 Cube manipulation

xcube.core.chunk.**chunk_dataset**(*dataset: [xarray.Dataset](), chunk_sizes: Optional[Dict[str, int]] = None, format_name: Optional[str] = None*) → [xarray.Dataset]()
Chunk *dataset* using *chunk_sizes* and optionally update encodings for given *format_name*.

Parameters

- **dataset** – input dataset

- **chunk_sizes** – mapping from dimension name to new chunk size

- **format_name** (`str`) – optional format, e.g. "zarr" or "netcdf4"

Returns the (re)chunked dataset

xcube.core.unchunk.**unchunk_dataset**(*dataset_path: str, var_names: Optional[Sequence[str]] = None, coords_only: bool = False*)
Unchunk dataset variables in-place.

Parameters

- **dataset_path** (`str`) – Path to ZARR dataset directory.

- **var_names** – Optional list of variable names.

- **coords_only** (`bool`) – Un-chunk coordinate variables only.

xcube.core.optimize.**optimize_dataset**(*input_path: str, output_path: Optional[str] = None, in_place: bool = False, unchunk_coords: Union[bool, str, Sequence[str]] = False, exception_type: Type[Exception] = <class 'ValueError'>*)
Optimize a dataset for faster access.

Reduces the number of metadata and coordinate data files in xcube dataset given by given by *dataset_path*. Consolidated cubes open much faster from remote locations, e.g. in object storage, because obviously much less HTTP requests are required to fetch initial cube meta information. That is, it merges all metadata files into a single top-level JSON file ".zmetadata".

If *unchunk_coords* is given, it also removes any chunking of coordinate variables so they comprise a single binary data file instead of one file per data chunk. The primary usage of this function is to optimize data cubes for cloud object storage. The function currently works only for data cubes using Zarr format. *unchunk_coords* can be a name, or list of names of the coordinate variable(s) to be consolidated. If boolean `True` is used, coordinate all variables will be consolidated.

Parameters

- **input_path** (`str`) – Path to input dataset with ZARR format.

- **output_path** (str) – Path to output dataset with ZARR format. May contain "{input}" template string, which is replaced by the input path's file name without file name extension.

- **in_place** (bool) – Whether to modify the dataset in place. If False, a copy is made and *output_path* must be given.

- **unchunk_coords** – The name of a coordinate variable or a list of coordinate variables whose chunks should be consolidated. Pass True to consolidate chunks of all coordinate variables.

- **exception_type** – Type of exception to be used on value errors.

5.6 Cube subsetting

5.7 Cube masking

class xcube.core.maskset.**MaskSet**(*flag_var: xarray.DataArray*)

A set of mask variables derived from a variable *flag_var* with the following CF attributes:

- One or both of *flag_masks* and *flag_values*

- *flag_meanings* (always required)

See https://cfconventions.org/Data/cf-conventions/cf-conventions-1.9/cf-conventions.html#flags for details on the use of these attributes.

Each mask is represented by an *xarray.DataArray*, has the name of the flag, is of type *numpy.unit8*, and has the dimensions of the given *flag_var*.

> **Parameters flag_var** – an *xarray.DataArray* that defines flag values. The CF attributes *flag_meanings* and one or both of *flag_masks* and *flag_values* are expected to exist and be valid.

classmethod get_mask_sets(*dataset: xarray.Dataset*) → Dict[str, *xcube.core.maskset.MaskSet*]

For each "flag" variable in given *dataset*, turn it into a MaskSet, store it in a dictionary.

> **Parameters dataset** – The dataset

> **Returns** A mapping of flag names to MaskSet. Will be empty if there

are no flag variables in *dataset*.

5.8 Rasterisation of Features

5.9 Cube metadata

xcube.core.edit.**edit_metadata**(*input_path: str, output_path: Optional[str] = None, metadata_path: Optional[str] = None, update_coords: bool = False, in_place: bool = False, monitor: Optional[Callable[[...], None]] = None, exception_type: Type[Exception] = <class 'ValueError'>*)

Edit the metadata of an xcube dataset.

Editing the metadata because it may be incorrect, inconsistent or incomplete. The metadata attributes should be given by a yaml file with the keywords to be edited. The function currently works only for data cubes using ZARR format.

> **Parameters**

- **input_path** (str) – Path to input dataset with ZARR format.

- **output_path** (str) – Path to output dataset with ZARR format. May contain "{input}" template string, which is replaced by the input path's file name without file name extentsion.

- **metadata_path** (str) – Path to the metadata file, which will edit the existing metadata.

- **update_coords** (bool) – Whether to update the metadata about the coordinates.

- **in_place** (bool) – Whether to modify the dataset in place. If False, a copy is made and *output_path* must be given.

- **monitor** – A progress monitor.

- **exception_type** – Type of exception to be used on value errors.

xcube.core.update.**update_dataset_attrs**(*dataset: xarray.Dataset*, *global_attrs: Optional[Dict[str, Any]] = None*, *update_existing: bool = False*, *in_place: bool = False*) → xarray.Dataset

Update spatio-temporal CF/THREDDS attributes given *dataset* according to spatio-temporal coordinate variables time, lat, and lon.

> **Parameters**
>
> - **dataset** – The dataset.
>
> - **global_attrs** – Optional global attributes.
>
> - **update_existing** (bool) – If True, any existing attributes will be updated.
>
> - **in_place** (bool) – If True, *dataset* will be modified in place and returned.
>
> **Returns** A new dataset, if *in_place* if False (default), else the passed and modified *dataset*.

xcube.core.update.**update_dataset_spatial_attrs**(*dataset: xarray.Dataset*, *update_existing: bool = False*, *in_place: bool = False*) → xarray.Dataset

Update spatial CF/THREDDS attributes of given *dataset*.

> **Parameters**
>
> - **dataset** – The dataset.
>
> - **update_existing** (bool) – If True, any existing attributes will be updated.
>
> - **in_place** (bool) – If True, *dataset* will be modified in place and returned.
>
> **Returns** A new dataset, if *in_place* if False (default), else the passed and modified *dataset*.

xcube.core.update.**update_dataset_temporal_attrs**(*dataset: xarray.Dataset*, *update_existing: bool = False*, *in_place: bool = False*) → xarray.Dataset

Update temporal CF/THREDDS attributes of given *dataset*.

> **Parameters**
>
> - **dataset** – The dataset.
>
> - **update_existing** (bool) – If True, any existing attributes will be updated.
>
> - **in_place** (bool) – If True, *dataset* will be modified in place and returned.
>
> **Returns** A new dataset, if *in_place* is False (default), else the passed and modified *dataset*.

5.10 Cube verification

5.11 Multi-resolution pyramids

5.12 Utilities

5.13 Plugin Development

class xcube.util.extension.**ExtensionRegistry**
 A registry of extensions. Typically used by plugins to register extensions.

 has_extension(*point: str*, *name: str*) → bool
  Test if an extension with given *point* and *name* is registered.

   Return type bool

   Parameters

   • **point** (str) – extension point identifier

   • **name** (str) – extension name

   Returns True, if extension exists

 get_extension(*point: str*, *name: str*) → Optional[*xcube.util.extension.Extension*]
  Get registered extension for given *point* and *name*.

   Parameters

   • **point** (str) – extension point identifier

   • **name** (str) – extension name

   Returns the extension or None, if no such exists

 get_component(*point: str*, *name: str*) → Any
  Get extension component for given *point* and *name*. Raises a ValueError if no such extension exists.

   Parameters

   • **point** (str) – extension point identifier

   • **name** (str) – extension name

   Returns extension component

 find_extensions(*point: str*, *predicate: Optional[Callable[[xcube.util.extension.Extension], bool]] =*
         None) → List[*xcube.util.extension.Extension*]
  Find extensions for *point* and optional filter function *predicate*.

  The filter function is called with an extension and should return a truth value to indicate a match or mismatch.

   Parameters

   • **point** (str) – extension point identifier

   • **predicate** – optional filter function

   Returns list of matching extensions

find_components(*point: str*, *predicate: Optional[Callable[[xcube.util.extension.Extension], bool]] = None*) → List[Any]

Find extension components for *point* and optional filter function *predicate*.

The filter function is called with an extension and should return a truth value to indicate a match or mismatch.

> **Parameters**
>
> - **point** (str) – extension point identifier
>
> - **predicate** – optional filter function
>
> **Returns** list of matching extension components

add_extension(*point: str*, *name: str*, *component: Optional[Any] = None*, *loader: Optional[Callable[[xcube.util.extension.Extension], Any]] = None*, ***metadata*) → *xcube.util.extension.Extension*

Register an extension *component* or an extension component *loader* for the given extension *point*, *name*, and additional *metadata*.

Either *component* or *loader* must be specified, but not both.

A given *loader* must be a callable with one positional argument *extension* of type `Extension` and is expected to return the actual extension component, which may be of any type. The *loader* will only be called once and only when the actual extension component is requested for the first time. Consider using the function `import_component()` to create a loader that lazily imports a component from a module and optionally executes it.

> **Return type** `Extension`
>
> **Parameters**
>
> - **point** (str) – extension point identifier
>
> - **name** (str) – extension name
>
> - **component** – extension component
>
> - **loader** – extension component loader function
>
> - **metadata** – extension metadata
>
> **Returns** a registered extension

remove_extension(*point: str*, *name: str*)

Remove registered extension *name* from given *point*.

> **Parameters**
>
> - **point** (str) – extension point identifier
>
> - **name** (str) – extension name

to_dict()

Get a JSON-serializable dictionary representation of this extension registry.

class xcube.util.extension.**Extension**(*point: str*, *name: str*, *component: Optional[Any] = None*, *loader: Optional[Callable[[xcube.util.extension.Extension], Any]] = None*, ***metadata*)

An extension that provides a component of any type.

Extensions are registered in a `ExtensionRegistry`.

Extension objects are not meant to be instantiated directly. Instead, `ExtensionRegistry.add_extension()` is used to register extensions.

Parameters

- **point** – extension point identifier

- **name** – extension name

- **component** – extension component

- **loader** – extension component loader function

- **metadata** – extension metadata

property is_lazy: bool
    Whether this is a lazy extension that uses a loader.

property component: Any
    Extension component.

property point: str
    Extension point identifier.

property name: str
    Extension name.

property metadata: Dict[str, Any]
    Extension metadata.

to_dict() → Dict[str, Any]
    Get a JSON-serializable dictionary representation of this extension.

xcube.util.extension.**import_component**(*spec: str*, *transform: Optional[Callable[[Any, xcube.util.extension.Extension], Any]] = None*, *call: bool = False*, *call_args: Optional[Sequence[Any]] = None*, *call_kwargs: Optional[Mapping[str, Any]] = None*) → Callable[[*xcube.util.extension.Extension*], Any]

Return a component loader that imports a module or module component from *spec*. To import a module, *spec* should be the fully qualified module name. To import a component, *spec* must also append the component name to the fully qualified module name separated by a color (":") character.

An optional *transform* callable my be used to transform the imported component. If given, a new component is computed:

```
component = transform(component, extension)
```

If the *call* flag is set, the component is expected to be a callable which will be called using the given *call_args* and *call_kwargs* to produce a new component:

```
component = component(*call_kwargs, **call_kwargs)
```

Finally, the component is returned.

Parameters

- **spec** (str) – String of the form "module_path" or "module_path:component_name"

- **transform** – callable that takes two positional arguments, the imported component and the extension of type `Extension`

- **call** (bool) – Whether to finally call the component with given *call_args* and *call_kwargs*

- **call_args** – arguments passed to a callable component if *call* flag is set

- **call_kwargs** – keyword arguments passed to callable component if *call* flag is set

> **Returns** a component loader

xcube.constants.**EXTENSION_POINT_INPUT_PROCESSORS** = **'xcube.core.gen.iproc'**
> The extension point identifier for input processor extensions

xcube.constants.**EXTENSION_POINT_DATASET_IOS** = **'xcube.core.dsio'**
> The extension point identifier for dataset I/O extensions

xcube.constants.**EXTENSION_POINT_CLI_COMMANDS** = **'xcube.cli'**
> The extension point identifier for CLI command extensions

xcube.util.plugin.**get_extension_registry**() → *xcube.util.extension.ExtensionRegistry*
> Get populated extension registry.

xcube.util.plugin.**get_plugins**() → Dict[str, Dict]
> Get mapping of "xcube_plugins" entry point names to JSON-serializable plugin meta-information.

WEB API AND SERVER

xcube's RESTful web API is used to publish data cubes to clients. Using the API, clients can

- List configured xcube datasets;

- Get xcube dataset details including metadata, coordinate data, and metadata about all included variables;

- Get cube data;

- Extract time-series statistics from any variable given any geometry;

- Get spatial image tiles from any variable;

- Get places (GeoJSON features including vector data) that can be associated with xcube datasets.

Later versions of API will also allow for xcube dataset management including generation, modification, and deletion of xcube datasets.

The complete description of all available functions is provided in the in the xcube Web API reference.

The web API is provided through the *xcube server* which is started using the *xcube serve* CLI command.

VIEWER APP

The xcube viewer app is a simple, single-page web application to be used with the xcube server.

7.1 Demo

To test the viewer app, you can use the xcube viewer demo. This is our Brockmann Consult Demo xcube viewer. Via the viewer's settings it is possible to change the xcube server url which is used for displaying data. To do so open the viewer's settings panels, select "Server". A "Select Server" panel is opened, click the "+" button to add a new server. Here are two demo servers that you may add for testing:

- DCS4COP Demo Server (`https://eodatabee-api.brockmann-consult.de/api`) providing ocean color variables in the North Sea area for the Data Cube Service for Copernicus (DCS4COP) EU project;

- Euro Data Cube Server (`https://edc-api.brockmann-consult.de/api`) has integrated amongst others a data cube with global essential climate variables (ECVs) variables from the ESA Earth System Data Lab Project. To access the Euro Data Cube viewer directly please visit https://edc-viewer.brockmann-consult.de .

7.2 Functionality

The xcube viewer functionality is described exemplary using the xcube viewer demo. The viewer visualizes data from the xcube datasets on top of a basemap. For zooming use the buttons in the top right corner of the map window or the zooming function of your computer mouse. A scale for the map is located in the lower right corner and in the upper left corner a corresponding legend to the mapped data of the data cube is available.
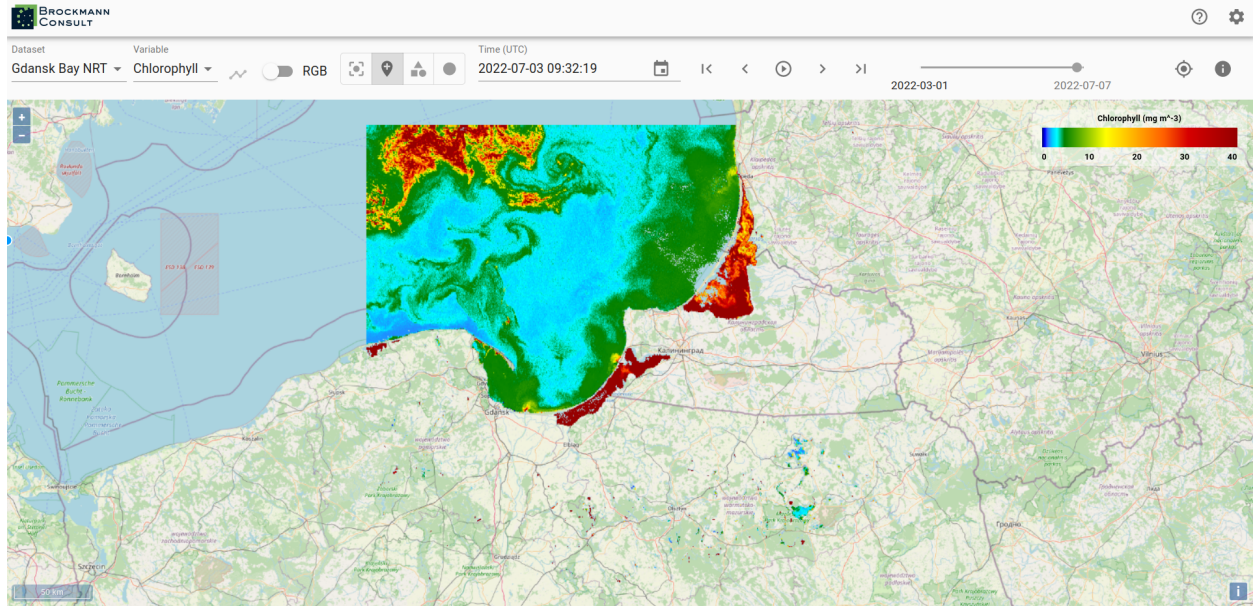
A xcube viewer may hold several xcube datasets which you can select via the drop-down menu *Dataset*. The viewed area automatically adjusts to a selected xcube dataset, meaning that if a newly selected dataset is located in a different region, the correct region is displayed on the map.
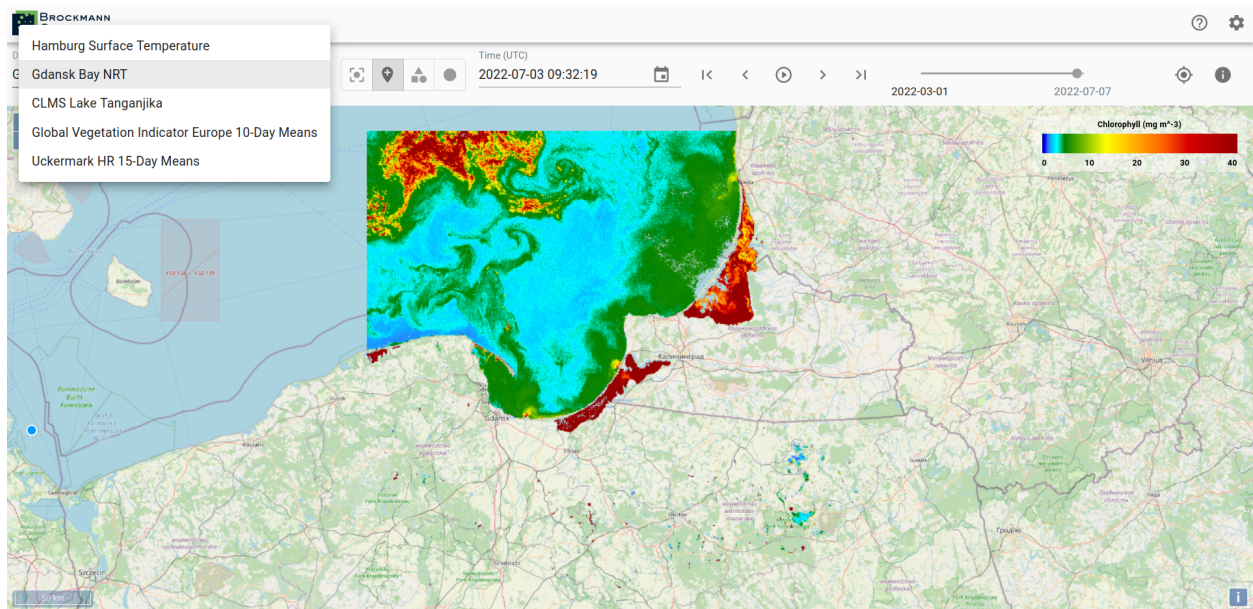
If more than one variable is available within a selected xcube dataset, you may change the variable by using the drop-down menu *Variable*.
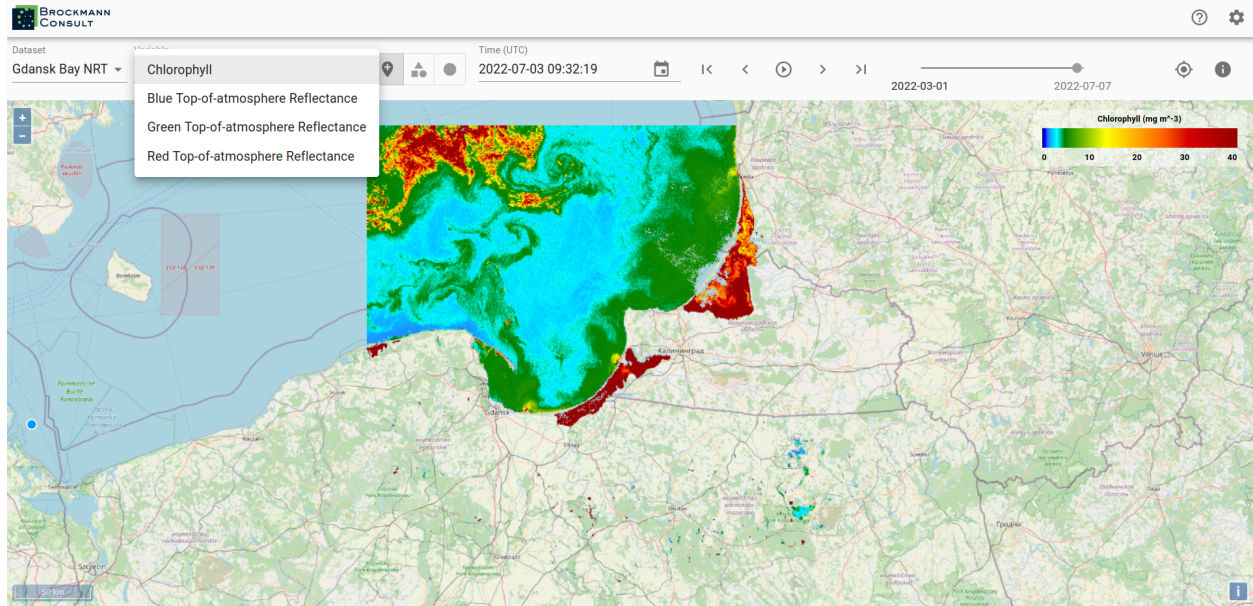
To see metadata for a dataset click on the *info*-icon on the right-hand side. Besides the dataset metadata you will see the metadata for the selected variable.
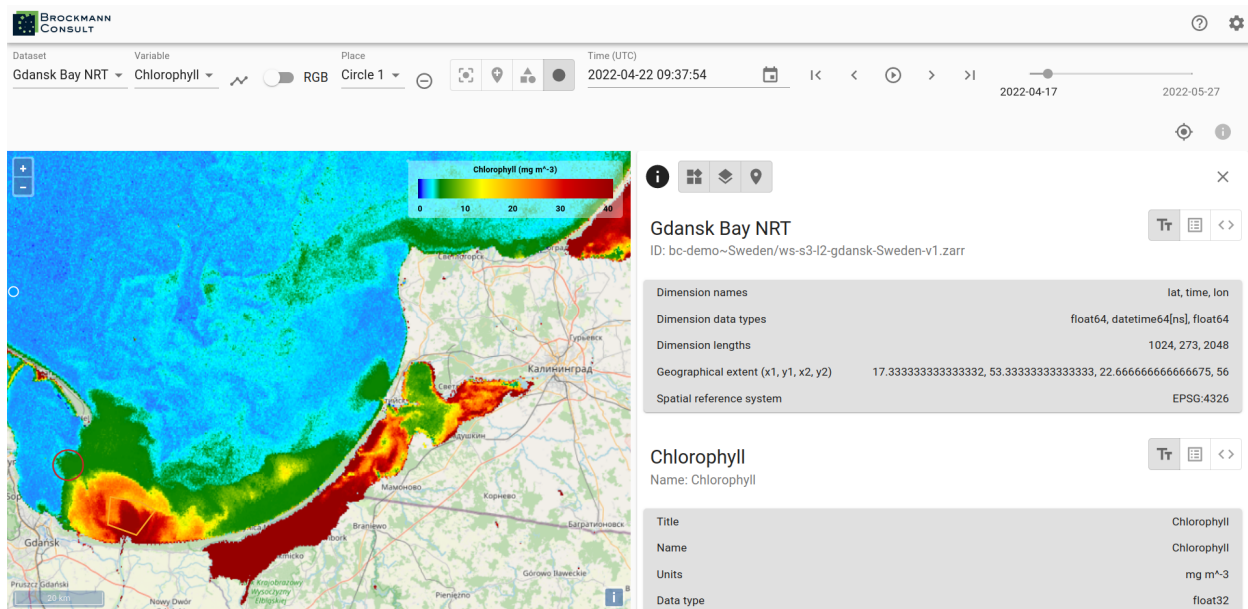
To obtain a time series set a point marker on the map and then select the *graph*-icon next to the *Variables* drop-down menu. You can select a different date by clicking into the time series graph on a value of interest. The data displayed in the viewer changes accordingly to the newly selected date.
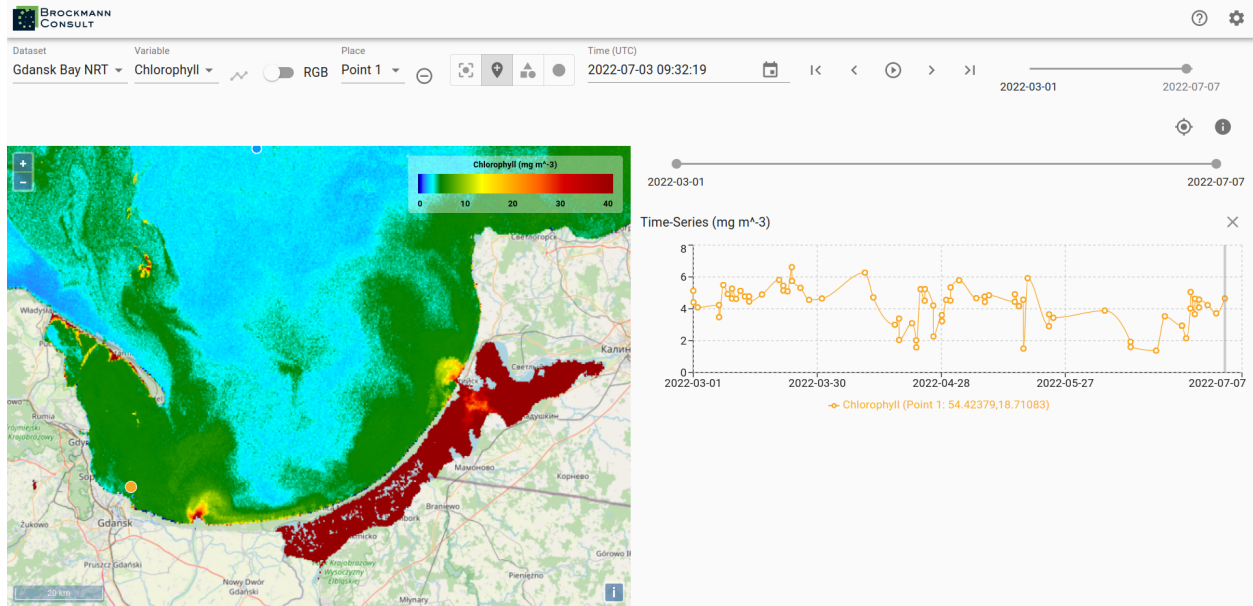
The current date is preserved when you select a different variable and the data of the variable is mapped for the date.
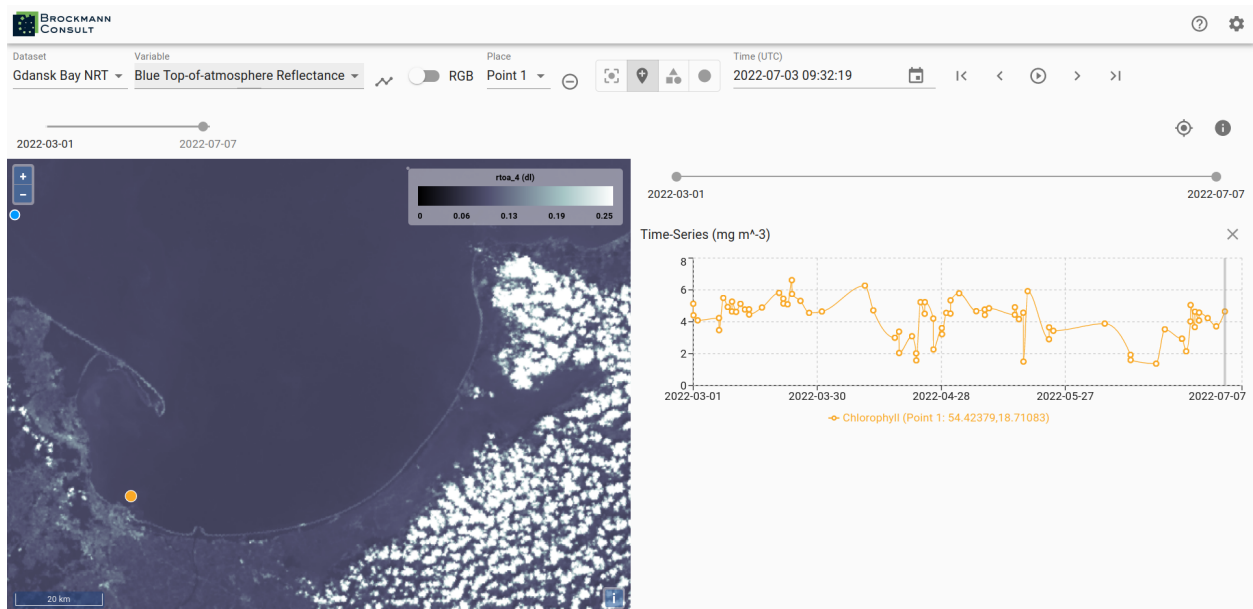
To generate a time series for the newly selected variable press the *time series*-icon again.
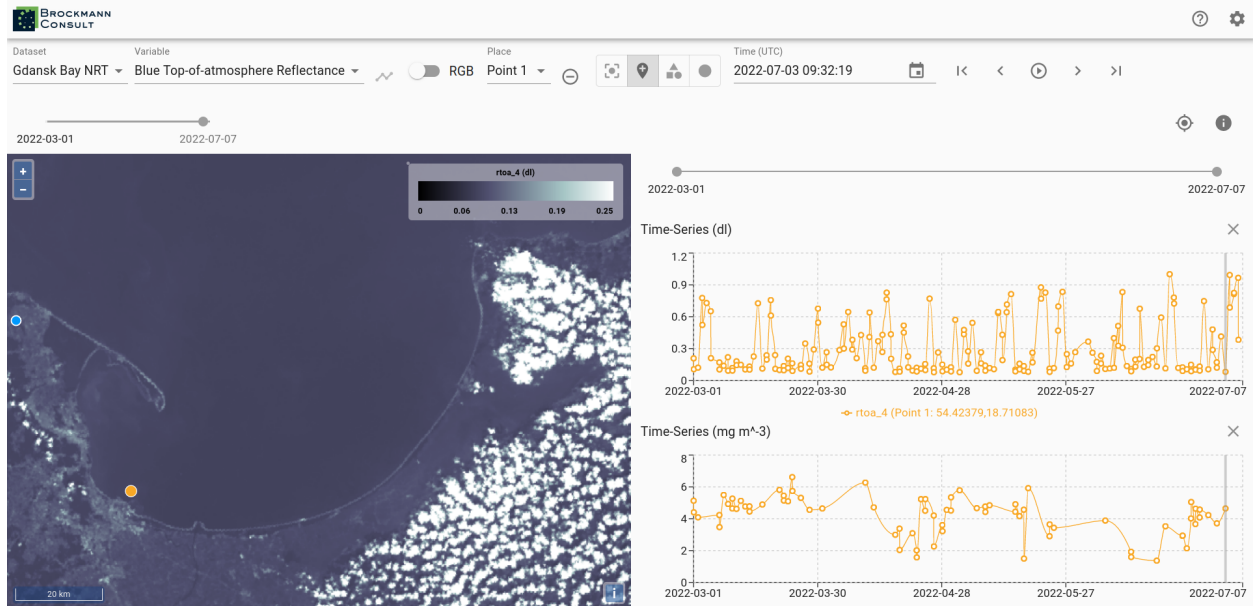
You may place multiple points on the map and you can generate time series for them. This allows a comparison between two locations. The color of the points corresponds to the color of the graph in the time series. You can find the coordinates of the point markers visualized in the time series beneath the graphs.
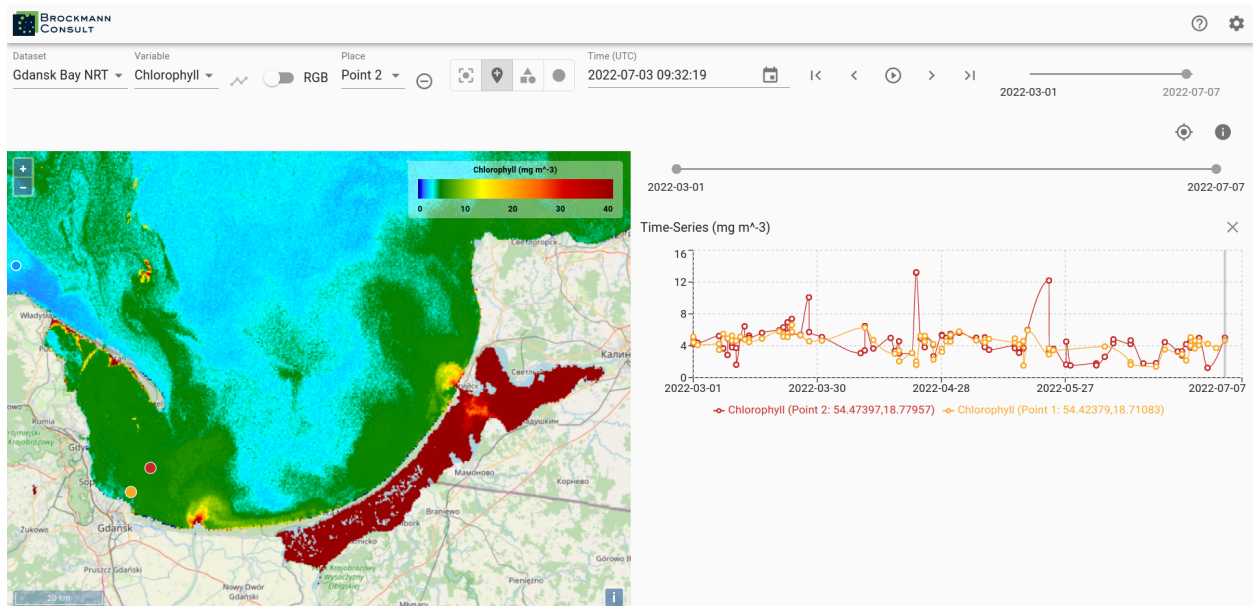
To delete a created location use the *remove*-icon next to the *Place* drop-down menu. Not only point location may be selected via the viewer, you can draw polygons and circular areas by using the icons on the right-hand side of the *Place* drop-down menu as well. You can visualize time series for areas, too.
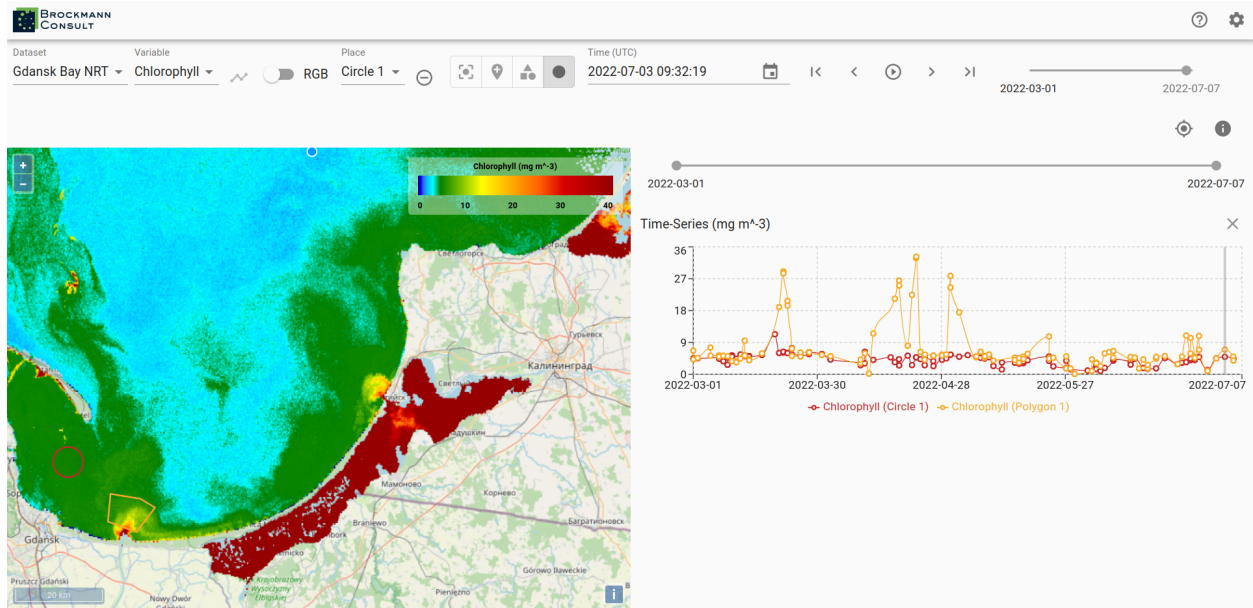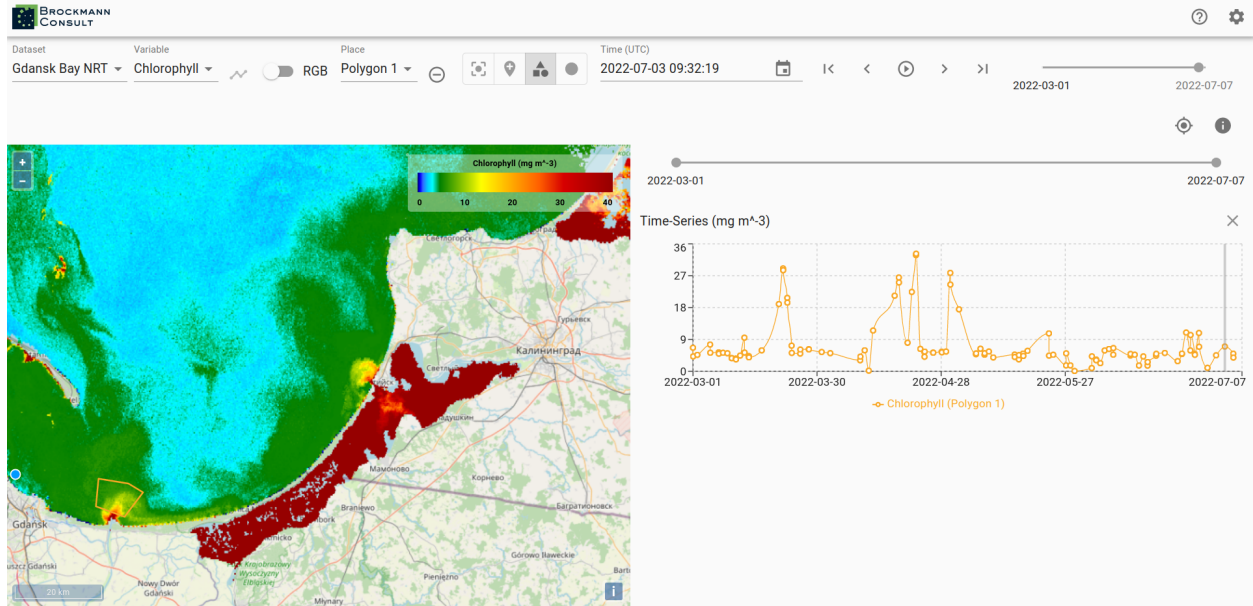
In order to change the date for the data display use the calendar or step through the time line with the arrows on the right-hand side of the calendar.
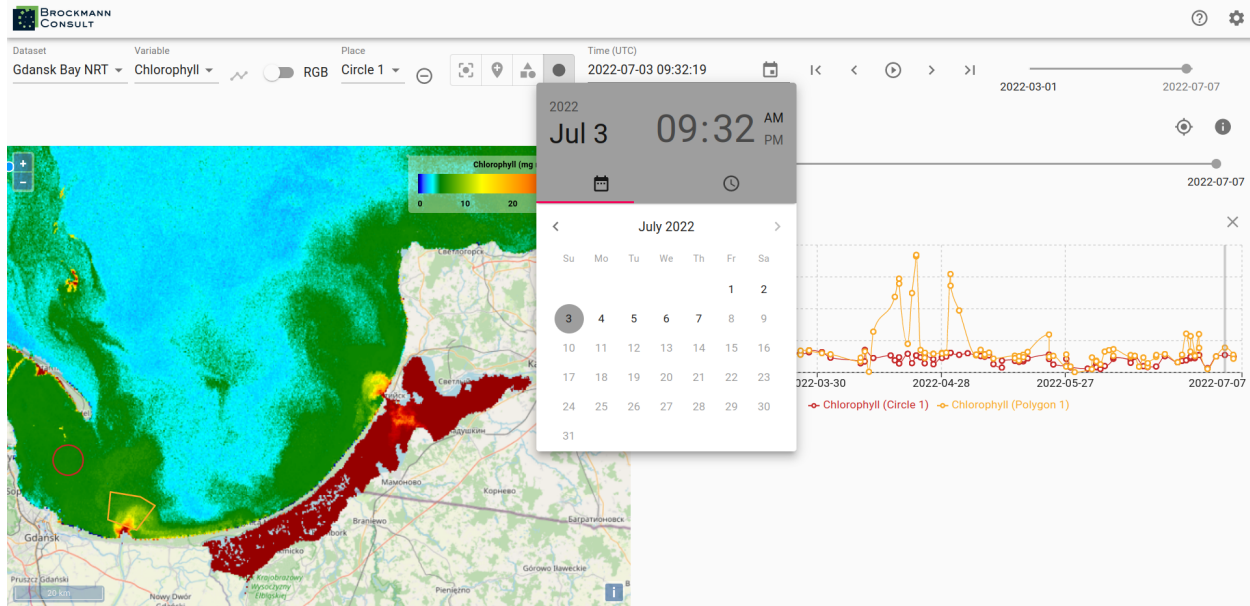
When a time series is displayed two time-line tools are visible, the upper one for selecting the date displayed on the map of the viewer and the lower one may be used to narrow the time frame displayed in the time series graph. Just above the graph of the time series on the right-hand side is an *x*-icon for removing the time series from the view and to left of it is an icon which sets the time series back to the whole time extent.
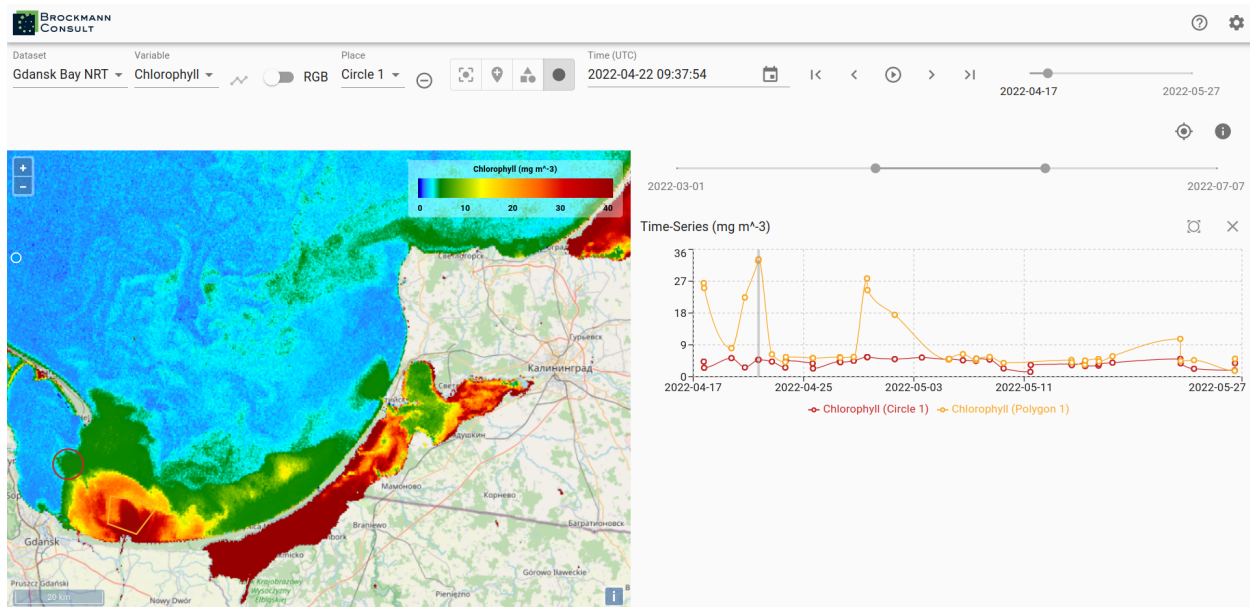
To adjust the default settings select the *Settings*-icon on the very top right corner. There you have the possibility to change the server url, in order to view data which is available via a different server. You can choose a different language - if available - as well as set your preferences of displaying data and graph of the time series.
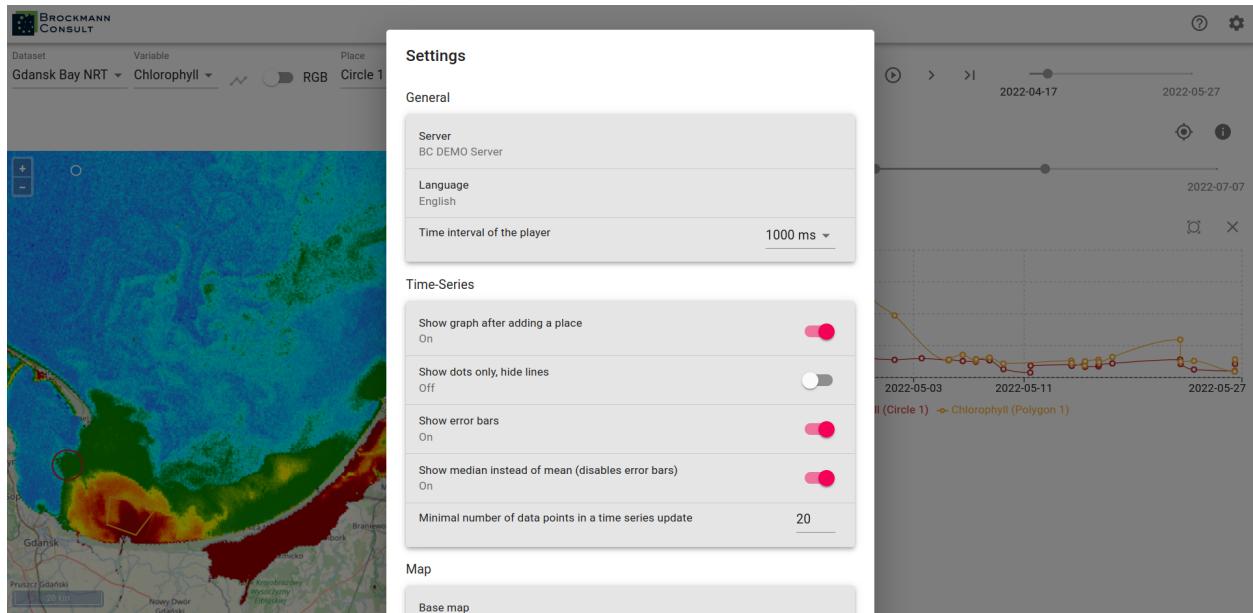
To see the map settings please scroll down in the settings window. There you can adjust the base map, switch the displayed projection between *Geographic* and *Mercator*. You can also choose to turn image smoothing on and to view the dataset boundaries.

On the very bottom of the *Settings* pop-up window you can see information about the viewer and server version.
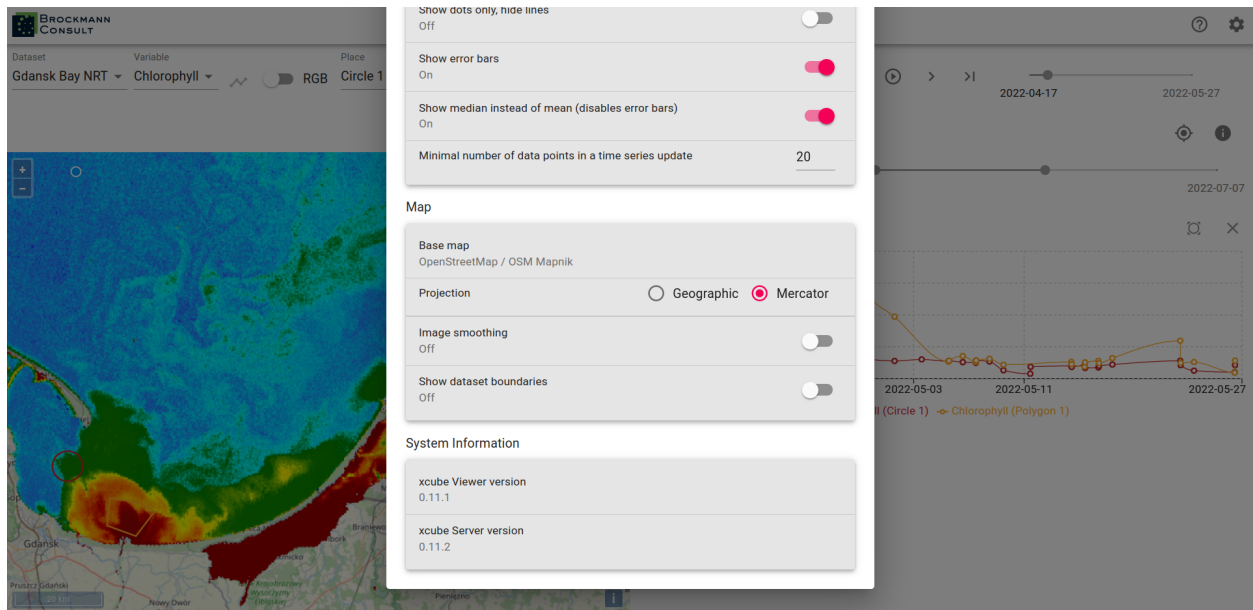
Back to the general view, if you would like to change the value ranges of the displayed variable you can do it by clicking into the area of the legend where the value ticks are located or you can enter the desired values in the *Minimum* and/or *Maximum* text field.
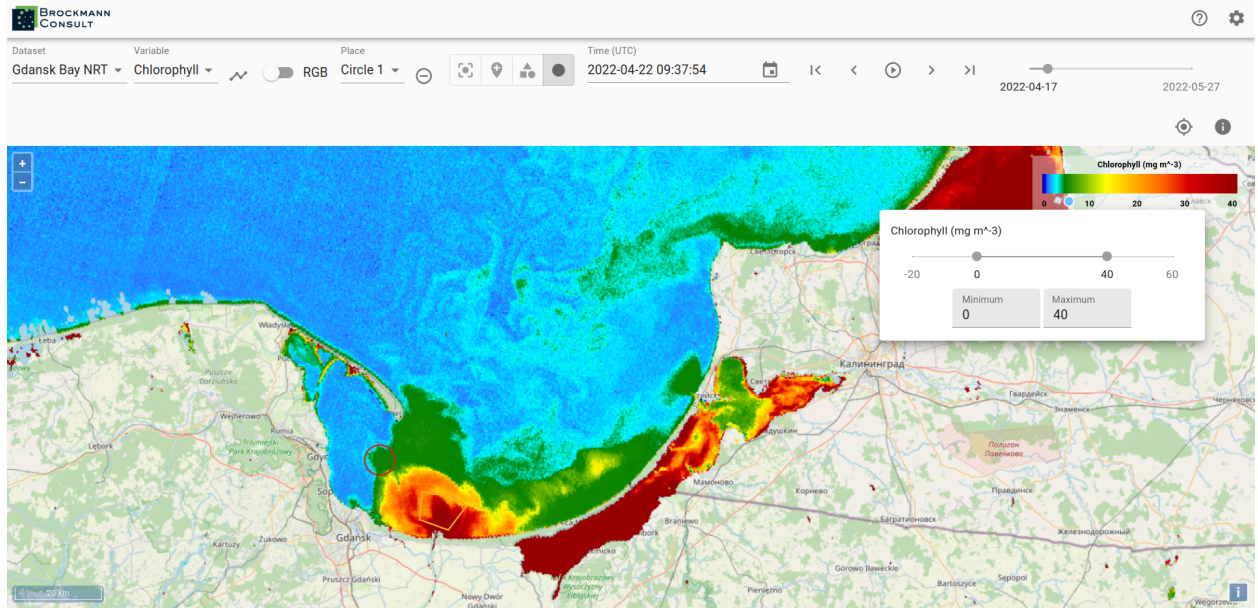
You can change the color mapping as well by clicking into the color range of the legend. There you can also decide to hide lower values and it is possible to adjust the opacity.
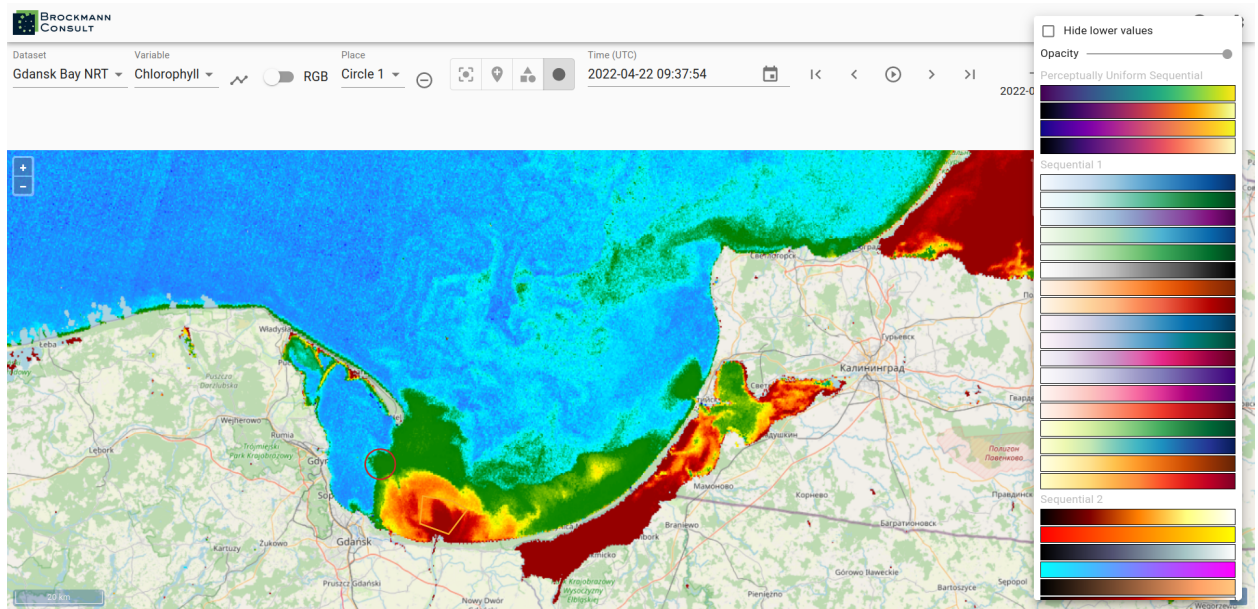
The xcube viewer app is constantly evolving and enhancements are added, therefore please be aware that the above described features may not always be completely up-to-date.

7.3 Build and Deploy

You can also build and deploy your own viewer instance. In the latter case, visit the xcube-viewer repository on GitHub and follow the instructions provides in the related README file.

THE XCUBE GENERATOR

8.1 Introduction

The *generator* is an xcube feature which allows users to create, manipulate, and write xcube datasets according to a supplied configuration. The same configuration can be used to generate a dataset on the user's local computer or remotely, using an online server.

The generator offers two main user interfaces: A Python API, configured using Python objects; and a command-line interface, configured using YAML or JSON files. The Python and file-based configurations have the same structure and are interconvertible.

The online generator service interfaces with the xcube client via a well-defined REST API; it is also possible for third-party clients to make use of this API directly, though it is expected that the Python and command-line interfaces will be more convenient in most cases.

8.2 Further documentation

This document aims to provide a brief overview of the generation process and the available configuration options. More details are available in other documents and in the code itself:

- Probably the most thorough documentation is available in the Jupyter demo notebooks in the xcube repository. These can be run in any JupyterLab environment containing an xcube installation. They combine explanation with interactive worked examples to demonstrate practical usage of the generator in typical use cases.

- For the Python API in particular, the xcube API documentation is generated from the docstrings included in the code itself and serves as a detailed low-level reference for individual Python classes and methods. The docstrings can also be read from a Python environment (e.g. using the ? postfix in IPython or JupyterLab) or, of course, by browsing the source code itself.

- For the YAML/JSON configuration syntax used with the command-line interface, there are several examples available in the examples/gen2/configs subdirectory of the xcube repository.

- For the REST API underlying the Python and command-line interfaces, there is a formal definition on Swagger-Hub, and one of the example notebooks demonstrates its usage with the Python *requests* library.

8.3 The generation process

The usual cube generation process is as follows:

1. The generator opens the input data store using the store identifier and parameters in the *input configuration*.

2. The generator reads from the input store the data specified in the *cube configuration* and uses them to create a data cube, often with additional manipulation steps such as resampling the data.

3. If an optional *code configuration* has been given, the user-supplied code is run on the created data cube, potentially modifying it.

4. The generator writes the generated cube to the data store specified in the *output configuration*.

8.4 Invoking the generator from a Python environment

The configurations for the various parts of the generator are used to initialize a `GeneratorRequest`, which is then passed to `xcube.core.gen2.generator.CubeGenerator.generate_cube`. The `generate_cube` method returns a *cube reference* which can be used to open the cube from the output data store.

The generator can also be directly invoked with a configuration file from a Python environment, using the `xcube.core.gen2.generator.CubeGenerator.from_file` method.

8.5 Invoking the generator from the command line

The generator can be invoked from the command line using the `xcube gen2` subcommand. (Note: the subcommand `xcube gen` invokes an earlier, deprecated generator feature which is not compatible with the generator framework described here.)

8.6 Configuration syntax

All Python configuration classes are defined in the `xcube.core.gen2` package, except for `CodeConfig`, which is in `xcube.core.byoa`.

The types in the parameter tables are given in an ad-hoc, semi-formal notation whose corresponding Python and JSON representations should be obvious. For the formal Python type definitions, see the signatures of the `__init__` methods of the configuration classes; for the formal JSON type definitions, see the JSON schemata (in JSON Schema format) produced by the `get_schema` methods of the configuration classes.

8.6.1 Remote generator service configuration

The command-line interface allows a *service configuration* for the remote generator service to be provided as a YAML or JSON file. This file defines the endpoint and access credentials for an online generator service. If it is provided, the specified remote service will be used to generate the cube. If it is omitted, the cube will be generated locally. The configuration file defines three values: `endpoint_url`, `client_id`, and `client_secret`. A typical service configuration YAML file might look as follows:

```
endpoint_url: "https://xcube-gen.brockmann-consult.de/api/v2/"
client_id: "93da366d7c39517865e4f141ddf1dd81"
client_secret: "d2l0aG91dCByZXN0cmljdGlvbiwgaW5jbHVkaW5nIHd"
```

8.6.2 Store configuration

In the command-line interface, an additional YAML or JSON file containing one or more *store configurations* may be supplied. A store configuration encapsulates a data store ID and an associated set of store parameters, which can then be referenced by an associated *store configuration identifier*. This identifier can be used in the input configuration, as described below. A typical YAML store configuration might look as follows:

```
sentinelhub_eu:
  title: SENTINEL Hub (Central Europe)
  description: Datasets from the SENTINEL Hub API deployment in Central Europe
  store_id: sentinelhub
  store_params:
    api_url: https://services.sentinel-hub.com
    client_id: myid123
    client_secret: 0c5892208a0a82f1599df026b5e19017

cds:
  title: C3S Climate Data Store (CDS)
  description: Selected datasets from the Copernicus CDS API
  store_id: cds
  store_params:
    normalize_names: true
    num_retries: 3

my_data_bucket:
  title: S3 output bucket
  description: An S3 bucket for output data sets
  store_id: s3
  store_params:
    root: cube-outputs
    storage_options:
      key: qwerty12345
      secret: 7ff889c0aea254d5e00440858289b85c
      client_kwargs:
        endpoint_url: https://my-endpoint.some-domain.org/
```

8.6.3 Input configuration

The input configuration defines the data store from which data for the cube are to be read, and any additional parameters which this data store requires.

The Python configuration object is `InputConfig`; the corresponding YAML configuration section is `input_configs`.

Parameter	Required?	Type	Description
`store_id`	N	str	Identifier for the data store
`opener_id`	N	str	Identifier for the data opener
`data_id`	Y	str	Identifier for the dataset
`store_params`	N	map(str→*)	Parameters for the data store
`open_params`	N	map(str→*)	Parameters for the data opener

`store_id` is a string identifier for a particular xcube data store, defined by the data store itself. If a store configuration file has been supplied (see above), a store configuration identifier can also be supplied here in place of a 'plain' store identifier. Store configuration identifiers must be prefixed by an @ symbol. If a store configuration identifier is supplied

in place of a store identifier, `store_params` values will be supplied from the predefined store configuration and can be omitted from the input configuration.

`data_id` is a string identifier for the dataset within a particular store.

The format and content of the `store_params` and `open_params` dictionaries is defined by the individual store or opener.

The generator service does not yet provide a remote interface to list available data stores, datasets, and store parameters (i.e. allowed values for the parameters in the table above). In a local xcube Python environment, you can list the currently available store identifiers with the expression `list(map(lambda e:  e.name, xcube.core.store.find_data_store_extensions()))`. You can create a local store object for an identifier `store_id` with `xcube.core.store.get_data_store_instance(store_id).store`. The store object provides methods `get_data_ids`, `get_data_store_params_schema`, and `get_open_data_params_schema` to describe the allowed values for the corresponding parameters. Note that the available stores and datasets on a remote xcube generator server may not be the same as those available in your local xcube environment.

8.6.4 Cube configuration

This configuration element defines the characteristics of the cube that should be generated. The Python configuration class is called `CubeConfig`, and the YAML section `cube_config`. All parameters are optional and will be filled in with defaults if omitted; the default values are dependent on the data store and dataset.

Parameter	Type	Units/Description
`variable_names`	[str, ...]	Available variables are data store dependent.
`crs`	str	PROJ string, JSON string with PROJ parameters, CRS WKT string, or authority string
`bbox`	[float, float, float, float]	Bounding-box (`min_x`, `min_y`, `max_x`, `max_y`) CRS-dependent, usually degrees
`spatial_res`	float or [float, float]	CRS-dependent, usually degrees
`tile_size`	int or [int, int]	pixels
`time_range`	str or [str, str]	ISO 8601 subset
`time_period`	str	integer + unit
`chunks`	map(str→null/int)	maps variable names to chunk sizes

The `crs` parameter string is interpreted using `` `CRS.from_string `` in the pyproj package <https://pyproj4.github.io/pyproj/dev/api/crs/crs.html#pyproj.crs.CRS.from_string>`__ and therefore accepts the same specifiers.

`time_range` specified the start and end of the requested time range. can be specified either as a date in the format `YYYY-MM-DD` or as a date and time in the format `YYYY-MM-DD HH:MM:SS`. If the time is omitted, it is taken to be `00:00:00` (the start of the day) for the start specifier and `24:00:00` (the end of the day) for the specifier. The end specifier may be omitted; in this case the current time is used.

`time_period` specified the duration of a single time step in the requested cube, which determines the temporal resolution. It consists of an integer denoting the number of time units, followed by single upper-case letter denoting the time unit. Valid time unit specifiers are D (day), W (week), M (month), and Y (year). Examples of `time_period` values: `1Y` (one year), `2M` (two months), `10D` (ten days).

The value of the `chunks` mapping determines how the generated data is chunked for storage. The chunking has no effect on the data itself, but can have a dramatic impact on data access speeds in different scenarios. The value of `chunks` is structured a map from variable names (corresponding to those specified by the `variable_names` parameter) to chunk sizes.

8.6.5 Code configuration

The code configuration supports multiple ways to define a *dataset processor* – fundamentally, a Python function which takes a dataset and returns a processed version of the input dataset. Since the code configuration can work directly with instantiated Python objects (which can't be stored in a YAML file), there are some differences in code configuration between the Python API and the YAML format.

Parameter	Type	Units/description
`_callable` †	Callable	Function to be called to process the datacube. Only available via Python API
`callable_ref`	str (non-empty)	A reference to a Python class or function, in the format `<module>:<function_or_class>`
`callable_params`	map(str→*)	Parameters to be passed to the specified callable
`inline_code` †	str (non-empty)	An inline snippet of Python code
`file_set` †	FileSet (Python) / map (YAML)	A bundle of Python modules or packages (see details below)
`install_required`	boolean	If set, indicates that `file_set` contains modules or packages to be installed.

All parameters are optional (as is the entire code configuration itself). The three parameters marked † are mutually exclusive: at most one of them may be given.

`_callable` provides the dataset processor directly and is only available in the Python API. It must be either a function or a class.

- If a function, it takes a `Dataset` and optional additional named parameters, and returns a `Dataset`. Any additional parameters are supplied in the `callable_params` parameter of the code configuration.

- If an object, it must implement a method `process_dataset`, which is treated like the function described above, and may optionally implement a class method `get_process_params_schema`, which returns a `JsonObjectSchema` describing the additional parameters. For convenience and clarity, the object may extend the abstract base class `DatasetProcessor`, which declares both these methods.

`callable_ref` is a string with the structure `<module>:<function_or_class>`, and specifies the function or class to call when `inline_code` or `file_set` is provided. The specified function or class is handled like the `_callable` parameter described above.

`callable_params` specifies a dictionary of named parameters which are passed to the processor function or method.

`inline_code` is a string containing Python source code. If supplied, it should contain the definition of a function or object as described for the `_callable` parameter. The module and class identifiers for the callable in the inline code snippet should be specified in `callable_ref` parameter.

`file_set` specifies a set of files which should be read from an fsspec file system and which contain a definition of a dataset processor. As with `inline_code`, the parameter `callable_ref` should also be supplied to tell the generator which class or function in the file set is the actual processor. The parameters of `file_set` are identical with those of the constructor of the corresponding Python `FileSet` class, and are as follows:

Parameter	Type	Description
`path`	str	fsspec-compatible root path specifier
`sub_path`	str	optional sub-path to append to main path
`includes`	[str]	include files matching any of these patterns
`excludes`	[str]	exclude files matching any of these patterns
`storage_params`	map(str→*)	FS-specific parameters (passed to fsspec)

8.6.6 Output configuration

This configuration element determines where the generated cube should be written to. The Python configuration class is called `OutputConfig`, and the YAML section `output_config`.

Parameter	Type	Units/description
`store_id`	str	Identifier of output store
`writer_id`	str	Identifier of data writer
`data_id`	str	Identifier under which to write the cube
`store_params`	map(str→*)	Store-dependent parameters for output store
`write_params`	map(str→*)	Writer-dependent parameters for output writer
`replace`	bool	If true, replace any existing data with the same identifier.

XCUBE DATASET SPECIFICATION

This document provides a technical specification of the protocol and format for *xcube datasets*, data cubes in the xcube sense.

The key words "MUST", "MUST NOT", "REQUIRED", "SHALL", "SHALL NOT", "SHOULD", "SHOULD NOT", "RECOMMENDED", "MAY", and "OPTIONAL" in this document are to be interpreted as described in RFC 2119.

9.1 Document Status

This is the latest version, which is still in development.

Version: 1.0, draft

Updated: 31.05.2018

9.2 Motivation

For many users of Earth observation data, multivariate coregistration, extraction, comparison, and analysis of different data sources is difficult, while data is provided in various formats and at different spatio-temporal resolutions.

9.3 High-level requirements

xcube datasets

- SHALL be time series of gridded, geo-spatial, geo-physical variables.
- SHALL use a common, equidistant, global or regional geo-spatial grid.
- SHALL shall be easy to read, write, process, generate.
- SHALL conform to the requirements of analysis ready data (ARD).
- SHALL be compatible with existing tools and APIs.
- SHALL conform to standards or common practices and follow a common data model.
- SHALL be formatted as self-contained datasets.
- SHALL be "cloud ready", in the sense that subsets of the data can be accessed by individual URIs.

ARD links:

- http://ceos.org/ard/

- https://landsat.usgs.gov/ard

- https://medium.com/planet-stories/analysis-ready-data-defined-5694f6f48815

9.4 xcube Dataset Schemas

9.4.1 Basic Schema

- Attributes metadata convention

    - SHALL be CF >= 1.7

    - SHOULD adhere to Attribute Convention for Data Discovery

- Dimensions:

    - SHALL be at least `time`, `bnds`, and MAY be any others.

    - SHALL all be greater than zero, but `bnds` must always be two.

- Temporal coordinate variables:

    - SHALL provide time coordinates for given time index.

    - MAY be non-equidistant or equidistant.

    - `time[time]` SHALL provide observation or average time of *cell centers*.

    - `time_bnds[time, bnds]` SHALL provide observation or integration time of *cell boundaries*.

    - Attributes:

        * Temporal coordinate variables MUST have `units`, `standard_name`, and any others.

        * `standard_name` MUST be `"time"`, `units` MUST have format `"<deltatime> since <datetime>"`, where `datetime` must have ISO-format. `calendar` may be given, if not, `"gregorian"` is assumed.

- Spatial coordinate variables

    - SHALL provide spatial coordinates for given spatial index.

    - SHALL be equidistant in either angular or metric units

- Cube variables:

    - SHALL provide *cube cells* with the dimensions as index.

    - SHALL have shape

        * `[time, ..., lat, lon]` (see WGS84 schema) or

        * `[time, ..., y, x]` (see Generic schema)

    - MAY have extra dimensions, e.g. `layer` (of the atmosphere), `band` (of a spectrum).

    - SHALL specify the `units` metadata attribute.

    - SHOULD specify metadata attributes that are used to identify missing values, namely `_FillValue` and / or `valid_min`, `valid_max`, see notes in CF conventions on these attributes.

    - MAY specify metadata attributes that can be used to visualise the data:

        * `color_bar_name`: Name of a predefined colour mapping. The colour bar is applied between a minimum and a maximum value.

* `color_value_min`, `color_value_max`: Minimum and maximum value for applying the colour bar. If not provided, minimum and maximum default to `valid_min`, `valid_max`. If neither are provided, minimum and maximum default to `0` and `1`.

9.4.2 WGS84 Schema (extends Basic)

- Dimensions:

    - SHALL be at least `time`, `lat`, `lon`, `bnds`, and MAY be any others.

- Spatial coordinate variables:

    - SHALL use WGS84 (EPSG:4326) CRS.

    - SHALL have `lat[lat]` that provides observation or average latitude of *cell centers* with attributes: `standard_name="latitude"` `units="degrees_north"`.

    - SHALL have `lon[lon]` that provides observation or average longitude of *cell centers* with attributes: `standard_name="longitude"` and `units="degrees_east"`.

    - SHOULD HAVE `lat_bnds[lat, bnds]`, `lon_bnds[lon, bnds]`: provide geodetic observation or integration coordinates of *cell boundaries*.

- Cube variables:

    - SHALL have shape `[time, ..., lat, lon]`.

9.4.3 Generic Schema (extends Basic)

- Dimensions: `time`, `y`, `x`, `bnds`, and any others.

    - SHALL be at least `time`, `y`, `x`, `bnds`, and MAY be any others.

- Spatial coordinate variables:

    - Any spatial grid and CRS.

    - `y[y]`, `x[x]`: provide spatial observation or average coordinates of *cell centers*.

        * Attributes: `standard_name`, `units`, other units describe the CRS / projections, see CF.

    - `y_bnds[y, bnds]`, `x_bnds[x, bnds]`: provide spatial observation or integration coordinates of *cell boundaries*.

    - MAY have `lat[y,x]`: latitude of *cell centers*.

        * Attributes: `standard_name="latitude"`, `units="degrees_north"`.

    - `lon[y,x]`: longitude of *cell centers*.

        * Attributes: `standard_name="longitude"`, `units="degrees_east"`.

- Cube variables:

    - MUST have shape `[time, ..., y, x]`.

9.5 xcube EO Processing Levels

This section provides an attempt to characterize xcube datasets generated from Earth Observation (EO) data according to their processing levels as they are commonly used in EO data processing.

9.5.1 Level-1C and Level-2C

- Generated from Level-1A, -1B, -2A, -2B EO data.
- Spatially resampled to common grid
  - Typically resampled at original resolution.
  - May be down-sampled: aggregation/integration.
  - May be upsampled: interpolation.
- No temporal aggregation/integration.
- Temporally non-equidistant.

9.5.2 Level-3

- Generated from Level-2C or -3 by temporal aggregation.
- No spatial processing.
- Temporally equidistant.
- Temporally integrated/aggregated.

XCUBE DEVELOPER GUIDE

Version 0.2, draft

IMPORTANT NOTE: Any changes to this doc must be reviewed by dev-team through pull requests.

10.1 Table of Contents

10.2 Versioning

We adhere to PEP-440. Therefore, the xcube software version uses the format `<major>.<minor>.<micro>` for released versions and `<major>.<minor>.<micro>.dev<n>` for versions in development.

- `<major>` is increased for major enhancements. CLI / API changes may introduce incompatibilities with former version.
- `<minor>` is increased for new features and and minor enhancements. CLI / API changes are backward compatible with former version.
- `<micro>` is increased for bug fixes and micro enhancements. CLI / API changes are backward compatible with former version.
- `<n>` is increased whenever the team (internally) deploys new builds of a development snapshot.

The current software version is in `xcube/version.py`.

10.3 Coding Style

We follow PEP-8, including its recommendation of PEP-484 syntax for type hints.

10.3.1 Updating code style in the existing codebase

A significant portion of the existing codebase does not adhere to our current code style guidelines. It is of course a goal to bring these parts into conformance with the style guide, but major style changes should not be bundled into pull requests focused on other improvements or bug fixes, because they obscure the significant code changes and make reviews difficult. Large-scale style and formatting updates should instead be made via dedicated pull requests.

10.3.2 Line length

As recommended in PEP-8, all lines should be limited to a maximum of 79 characters, including docstrings and comments.

10.3.3 Quotation marks for string literals

In general, single quotation marks should always be used for string literals. Double quotation marks should only be used if there is a compelling reason to do so in a particular case.

10.4 Main Packages

- `xcube.core` - Hosts core API functions. Code in here should be maintained w.r.t. backward compatibility. Therefore think twice before adding new or change existing core API.

- `xcube.cli` - Hosts CLI commands. CLI command implementations should be lightweight. Move implementation code either into `core` or `util`.CLI commands must be maintained w.r.t. backward compatibility. Therefore think twice before adding new or change existing CLI commands.

- `xcube.webapi` - Hosts Web API functions. Web API command implementations should be lightweight. Move implementation code either into `core` or `util`.Web API interface must be maintained w.r.t. backward compatibility. Therefore think twice before adding new or change existing web API.

- `xcube.util` - Mainly implementation helpers. Comprises classes and functions that are used by `cli`, `core`, `webapi` in order to maximize modularisation and testability but to minimize code duplication.The code in here must not be dependent on any of `cli`, `core`, `webapi`. The code in here may change often and in any way as desired by code implementing the `cli`, `core`, `webapi` packages.

The following sections will guide you through extending or changing the main packages that form xcube's public interface.

10.4.1 Package `xcube.cli`

Checklist

Make sure your change

1. is covered by unit-tests (package `test/cli`);

2. is reflected by the CLI's doc-strings and tools documentation (currently in `README.md`);

3. follows existing xcube CLI conventions;

4. follows PEP8 conventions;

5. is reflected in API and WebAPI, if desired;

6. is reflected in `CHANGES.md`.

Hints

Make sure your new CLI command is in line with the others commands regarding command name, option names, as well as metavar arguments names. The CLI command name shall ideally be a verb.

Avoid introducing new option arguments if similar options are already in use for existing commands.

In the following common arguments and options are listed.

Input argument:

```
@click.argument('input')
```

If input argument is restricted to an xcube dataset:

```
@click.argument('cube')
```

Output (directory) option:

```
@click.option('--output', '-o', metavar='OUTPUT',
              help='Output directory. If omitted, "INPUT.levels" will be used.')
```

Output format:

```
@click.option('--format', '-f', metavar='FORMAT', type=click.Choice(['zarr', 'netcdf']),
              help="Format of the output. If not given, guessed from OUTPUT.")
```

Output parameters:

```
@click.option('--param', '-p', metavar='PARAM', multiple=True,
              help="Parameter specific for the output format. Multiple allowed.")
```

Variable names:

```
@click.option('--variable',--var', metavar='VARIABLE', multiple=True,
              help="Name of a variable. Multiple allowed.")
```

For parsing CLI inputs, use helper functions that are already in use. In the CLI command implementation code, raise `click.ClickException(message)` with a clear `message` for users.

Common xcube CLI options like `-f` for FORMAT should be lower case letters and specific xcube CLI options like `-S` for SIZE in `xcube gen` are recommended to be uppercase letters.

Extensively validate CLI inputs to avoid that API functions raise `ValueError`, `TypeError`, etc. Such errors and their message texts are usually hard to understand by users. They are actually dedicated to to developers, not CLI users.

There is a global option `--traceback` flag that user can set to dump stack traces. You don't need to print stack traces from your code.

10.4.2 Package `xcube.core`

Checklist

Make sure your change

1. is covered by unit-tests (package `test/core`);

2. is covered by API documentation;

3. follows existing xcube API conventions;

4. follows PEP8 conventions;

5. is reflected in xarray extension class `xcube.core.xarray.DatasetAccessor`;

6. is reflected in CLI and WebAPI if desired;

7. is reflected in `CHANGES.md`.

Hints

Create new module in `xcube.core` and add your functions. For any functions added make sure naming is in line with other API. Add clear doc-string to the new API. Use Sphinx RST format.

Decide if your API methods requires *xcube datasets* as inputs, if so, name the primary dataset argument `cube` and add a keyword parameter `cube_asserted:  bool = False`. Otherwise name the primary dataset argument `dataset`.

Reflect the fact, that a certain API method or function operates only on datasets that conform with the xcube dataset specifications by using `cube` in its name rather than `dataset`. For example `compute_dataset` can operate on any xarray datasets, while `get_cube_values_for_points` expects a xcube dataset as input or `read_cube` ensures it will return valid xcube datasets only.

In the implementation, if `not cube_asserted`, we must assert and verify the `cube` is a cube. Pass `True` to `cube_asserted` argument of other API called later on:

```
from xcube.core.verify import assert_cube

def frombosify_cube(cube: xr.Dataset, ..., cube_asserted: bool = False):
    if not cube_asserted:
        assert_cube(cube)
    ...
    result = bibosify_cube(cube, ..., cube_asserted=True)
    ...
```

If `import xcube.core.xarray` is imported in client code, any `xarray.Dataset` object will have an extra property `xcube` whose interface is defined by the class `xcube.core.xarray.DatasetAccessor`. This class is an xarray extension that is used to reflect `xcube.core` functions and make it directly applicable to the `xarray.Dataset` object.

Therefore any xcube API shall be reflected in this extension class.

10.4.3 Package `xcube.webapi`

Checklist

Make sure your change

1. is covered by unit-tests (package `test/webapi`);

2. is covered by Web API specification and documentation (currently in `webapi/res/openapi.yml`);

3. follows existing xcube Web API conventions;

4. follows PEP8 conventions;

5. is reflected in CLI and API, if desired;

6. is reflected in `CHANGES.md`.

10.4.4 Hints

- The Web API is defined in `webapi.app` which defines mapping from resource URLs to handlers

- All handlers are implemented in `webapi.handlers`. Handler code just delegates to dedicated controllers.

- All controllers are implemented in `webapi.controllers.*`. They might further delegate into `core.*`

10.5 Development Process

1. Make sure there is an issue ticket for your code change work item

2. Select issue, priorities are as follows

    1. "urgent" and ("important" and "bug")

    2. "urgent" and ("important" or "bug")

    3. "urgent"

    4. "important" and "bug"

    5. "important" or "bug"

    6. others

3. Make sure issue is assigned to you, if unclear agree with team first.

4. Add issue label "in progress".

5. Create development branch named `"<developer>-<issue>-<title>"` (see *below*).

6. Develop, having in mind the checklists and implementation hints above.

    1. In your first commit, refer the issue so it will appear as link in the issue history

    2. Develop, test, and push to the remote branch as desired.

    3. In your last commit, utilize checklists above. (You can include the line "closes #`<issue>`" in your commit message to auto-close the issue once the PR is merged.)

7. Create PR if build servers succeed on your branch. If not, fix issue first.For the PR assign the team for review, agree who is to merge. Also reviewers should have checklist in mind.

8. Merge PR after all reviewers are accepted your change. Otherwise go back.

9. Remove issue label "in progress".

10. Delete the development branch.

11. If the PR is only partly solving an issue:

    1. Make sure the issue contains a to-do list (checkboxes) to complete the issue.

    2. Do not include the line "closes #`<issue>`" in your last commit message.

    3. Add "relates to issue#" in PR.

    4. Make sure to check the corresponding to-do items (checkboxes) *after* the PR is merged.

    5. Remove issue label "in progress".

    6. Leave issue open.

10.6 Branches and Releases

10.6.1 Target Branch

The `master` branch contains latest developments, including new features and fixes. Its software version string is always `<major>.<minor>.<micro>.dev<n>`. The branch is used to generate major, minor, or maintenance releases. That is, either `<major>`, `<minor>`, or `<fix>` is increased. Before a release, the last thing we do is to remove the `.dev<n>` suffix, after a release, the first thing we do is to increase the `micro` version and add the `.dev<n>` suffix.

10.6.2 Development Branches

Development branches should be named `<developer>-<issue>-<title>` where

- `<developer>` is the github name of the code author
- `<issue>` is the number of the issue in the github issue tracker that is targeted by the works on this branch
- `<title>` is either the name of the issue or an abbreviated version of it

10.7 Release Process

10.7.1 Release on GitHub

This describes the release process for `xcube`. For a plugin release, you need to adjust the paths accordingly.

- Check issues in progress, close any open issues that have been fixed.
- Make sure that all unit tests pass and that test coverage is 100% (or as near to 100% as practicable).
- In `xcube/version.py` remove the `.dev` suffix from version name.
- Adjust version in `Dockerfile` accordingly.
- Make sure `CHANGES.md` is complete. Remove the suffix `(in development)` from the last version headline.
- Push changes to either master or a new maintenance branch (see above).
- Await results from Travis CI and ReadTheDocs builds. If broken, fix.
- Go to xcube/releases and press button "Draft a new Release".

    – Tag version is: `v${version}` (with a "v" prefix)

    – Release title is: `${version}` (without a "v" prefix)

    – Paste latest changes from `CHANGES.md` into field "Describe this release"

    – Press "Publish release" button

- After the release on GitHub, rebase sources, if the branch was `master`, create a new maintenance branch (see above)

- In `xcube/version.py` increase version number and append a `.dev0` suffix to the version name so that it is still PEP-440 compatible.

- Adjust version in `Dockerfile` accordingly.

- In `CHANGES.md` add a new version headline and attach `(in development)` to it.

- Push changes to either master or a new maintenance branch (see above).

- Activate new doc version on ReadTheDocs.

Go through the same procedure for all xcube plugin packages dependent on this version of xcube.

10.7.2 Release on Conda-Forge

These instructions are based on the documentation at conda-forge.

Conda-forge packages are produced from a github feedstock repository belonging to the conda-forge organization. A repository's feedstock is usually located at `https://github.com/conda-forge/<repo-name>-feedstock`, e.g., https://github.com/conda-forge/xcube-feedstock. The package is updated by

- forking the repository

- creating a new branch for the changes

- creating a pull request to merge this branch into conda-forge's feedstock repository (this is done automatically if the build number is 0).

The first of these steps is usually already done. You may find forks at `https://github.com/dcs4cop/<repo-name>-feedstock` .

In detail, the steps are:

1. Update the dcs4cop fork of the feedstock repository, if it's not already up to date with conda-forge's upstream repository.

2. Clone the repository locally and create a new branch. The name of the branch is not strictly prescribed, but it's sensible to choose an informative name like `update_0_5_3`.

3. In case the build number is 0, a bot will render the feedstock during the pull request. Otherwise, conduct the following steps: Rerender the feedstock using conda-smithy. This updates common conda-forge feedstock files. It's probably easiest to install conda-smithy in a fresh environment for this:

   `conda install -c conda-forge conda-smithy`

   `conda smithy rerender -c auto`

4. Update `recipe/meta.yaml` for the new version. Mainly this will involve the following steps:

   1. Update the value of the version variable (or, if the version number has not changed, increment the build number).

   2. If the version number has changed, ensure that the build number is set to 0.

   3. Update the sha256 hash of the source archive prepared by GitHub.

4. If the dependencies have changed, update the list of dependencies in the `-run` subsection to match those in the environment.yml file.

5. Commit the changes and push them to GitHub. A pull request at the feedstock repository on conda-forge will be automatically created by a bot if the build number is 0. If it is higher, you will have to create the pull request yourself.

6. Once conda-forge's automated checks have passed, merge the pull request.

7. Merge the newly-merged changes from the master branch on conda-forge back to the master branch of the dcs4cop fork. This step is not necessarily needed for the release, but it helps to avoid messy parallel branches.

Once the pull request has been merged, the updated package should usually become available from conda-forge within a couple of hours.

TODO: Describe deployment of xcube Docker image after release

If any changes apply to `xcube serve` and the xcube Web API:

Make sure changes are reflected in `xcube/webapi/res/openapi.yml`. If there are changes, sync `xcube/webapi/res/openapi.yml` with xcube Web API docs on SwaggerHub.

Check if changes affect the xcube-viewer code. If so make sure changes are reflected in xcube-viewer code and test viewer with latest xcube Web API. Then release a new xcube viewer.

10.7.3 xcube Viewer

- Cd into viewer project directory (`.../xcube-viewer/.`).
- Remove the `-dev` suffix from `version` property in `package.json`.
- Remove the `-dev` suffix from `version` constant in `src/config.ts`.
- Make sure `CHANGES.md` is complete. Remove the suffix `(in development)` from the last version headline.
- Build the app and test the build using a local http-server, e.g.:

  $ npm install -g http-server $ cd build $ http-server -p 3000 -c-1

- Push changes to either master or a new maintenance branch (see above).
- Goto [xcube-viewer/releases](#) and press button "Draft a new Release".

  - Tag version is: `v${version}` (with a "v" prefix).
  - Release title is: `${version}`.
  - Paste latest changes from `CHANGES.md` into field "Describe this release".
  - Press "Publish release" button.

- After the release on GitHub, if the branch was `master`, create a new maintenance branch (see above).
- Increase `version` property and `version` constant in `package.json` and `src/config.ts` and append `-dev.0` suffix to version name so it is SemVer compatible.
- In `CHANGES.md` add a new version headline and attach `(in development)` to it.
- Push changes to either master or a new maintenance branch (see above).
- Deploy builds of `master` branches to related web content providers.

PLUGINS

xcube's functionality can be extended by plugins. A plugin contributes extensions to specific extension points defined by xcube. Plugins are detected and dynamically loaded, once the available extensions need to be inquired.

11.1 Installing Plugins

Plugins are installed by simply installing the plugin's package into xcube's Python environment.

In order to be detected by xcube, an plugin package's name must either start with `xcube_` or the plugin package's `setup.py` file must specify an entry point in the group `xcube_plugins`. Details are provided below in section *plugin_development*.

11.2 Available Plugins

11.2.1 SENTINEL Hub

The xcube_sh plugin adds support for the SENTINEL Hub Cloud API. It extends xcube by a new Python API function `xcube_sh.cube.open_cube` to create data cubes from SENTINEL Hub on-the-fly. It also adds a new CLI command `xcube sh gen` to generate and write data cubes created from SENTINEL Hub into the file system.

11.2.2 ESA CCI Open Data Portal

The xcube_cci plugin provides support for the ESA CCI Open Data Portal.

11.2.3 Copernicus Climate Data Store

The xcube_cds plugin provides support for the Copernicus Climate Data Store.

xcube, Release 0.12.1

11.2.4 Cube Generation

xcube's GitHub organisation currently hosts a few plugins that add new *input processor* extensions (see below) to xcube's data cube generation tool *xcube gen*. They are very specific but are a good starting point for developing your own input processors:

- xcube_gen_bc - adds new input processors for specific Ocean Colour Earth Observation products derived from the Sentinel-3 OLCI measurements.

- xcube_gen_rbins - adds new input processors for specific Ocean Colour Earth Observation products derived from the SEVIRI measurements.

- xcube_gen_vito - adds new input processors for specific Ocean Colour Earth Observation products derived from the Sentinel-2 MSI measurements.

11.3 Plugin Development

11.3.1 Plugin Definition

An xcube plugin is a Python package that is installed in xcube's Python environment. xcube can detect plugins either

1. by naming convention (more simple);

2. by entry point (more flexible).

By naming convention: Any Python package named `xcube_<name>` that defines a plugin *initializer function* named `init_plugin` either defined in `xcube_<name>/plugin.py` (preferred) or `xcube_<name>/__init__.py` is an xcube plugin.

By entry point: Any Python package installed using Setuptools that defines a non-empty entry point group `xcube_plugins` is an xcube plugin. An entry point in the `xcube_plugins` group has the format `<name> = <fully-qualified-module-path>:<init-func-name>`, and therefore specifies where plugin *initializer function* named `<init-func-name>` is found. As an example, refer to the xcube standard plugin definitions in xcube's setup.py file.

For more information on Setuptools entry points refer to section Creating and discovering plugins in the Python Packing User Guide and Dynamic Discovery of Services and Plugins in the Setuptools documentation.

11.3.2 Initializer Function

xcube plugins are initialized using a dedicated function that has a single *extension registry* argument of type `xcube.util.extension.ExtensionRegistry`, that is used by plugins's to register their extensions to xcube. By convention, this function is called `init_plugin`, however, when using entry points, it can have any name. As an example, here is the initializer function of the SENTINEL Hub plugin `xcube_sh/plugin.py`::

```python
from xcube.constants import EXTENSION_POINT_CLI_COMMANDS
from xcube.util import extension


def init_plugin(ext_registry: extension.ExtensionRegistry):
    """xcube SentinelHub extensions"""
    ext_registry.add_extension(loader=extension.import_component('xcube_sh.cli:cli'),
                               point=EXTENSION_POINT_CLI_COMMANDS,
                               name='sh_cli')
```

92 **Chapter 11. Plugins**

11.3.3 Extension Points and Extensions

When a plugin is loaded, it adds its extensions to predefined *extension points* defined by xcube. xcube defines the following extension points:

- `xcube.core.gen.iproc`: input processor extensions

- `xcube.core.dsio`: dataset I/O extensions

- `xcube.cli`: Command-line interface (CLI) extensions

An extension is added to the extension registry's `add_extension` method. The extension registry is passed to the plugin initializer function as its only argument.

11.3.4 Input Processor Extensions

Input processors are used the `xcube gen` CLI command and `gen_cube` API function. An input processor is responsible for processing individual time slices after they have been opened from their sources and before they are appended to or inserted into the data cube to be generated. New input processors are usually programmed to support the characteristics of specific `xcube gen` inputs, mostly specific Earth Observation data products.

By default, xcube uses a standard input processor named `default` that expects inputs to be individual NetCDF files that conform to the CF-convention. Every file is expected to contain a single spatial image with dimensions `lat` and `lon` and the time is expected to be given as global attributes.

If your input files do not conform with the `default` expectations, you can extend xcube and write your own input processor. An input processor is an implementation of the `xcube.core.gen.iproc.InputProcessor` or `xcube.core.gen.iproc.XYInputProcessor` class.

As an example take a look at the implementation of the `default` input processor xcube.core.gen.iproc.DefaultInputProcessor or the various input processor plugins mentioned above.

The extension point identifier is defined by the constant `xcube.constants.EXTENSION_POINT_INPUT_PROCESSORS`.

11.3.5 Dataset I/O Extensions

More coming soon. . .

The extension point identifier is defined by the constant `xcube.constants.EXTENSION_POINT_DATASET_IOS`.

11.3.6 CLI Extensions

CLI extensions enhance the `xcube` command-line tool by new sub-commands. The xcube CLI is implemented using the click library, therefore the extension components must be click commands or command groups.

The extension point identifier is defined by the constant `xcube.constants.EXTENSION_POINT_CLI_COMMANDS`.

TWELVE

INDICES AND TABLES

- genindex
- modindex
- search