BROCKMANN CONSULT

Turning Environmental Data Into Knowledge

Application packages in the xcube ecosystem

Pontus Lurcock, Brockmann Consult

4 September 2025

Requirements (you should have these already!)

See the course website at
https://xcube-dev.github.io/summerschool25/
for full details.

- Your preferred **terminal emulator**, preferably running the bash shell.
- A conda-based **Python environment**, which we'll use to set up an environment in which to run the Python tools. I recommend mamba. Installation instructions here.
- git
- docker

Overview of this course

In this course, you'll get to know:

- OGC Earth Observation **Application Packages**, a framework that lets you package almost any software into a reusable module for deployment on cloud infrastructure.

- **xcube**, a multitalented, ever-growing toolkit and ecosystem for working with data cubes in Python.

- **xcengine**, a new tool that lets you automatically turn your Python Jupyter notebooks into both Application Packages and xcube server containers.

Course structure

For each part of the course, I'll first present a few slides from this deck to introduce the topic and give background information.

Then we'll move to the hands-on section of that part. In the GitHub repository is a notebook called **summerschool.ipynb** which will guide you through the hands-on parts.

Part 1: Introducing Application Packages

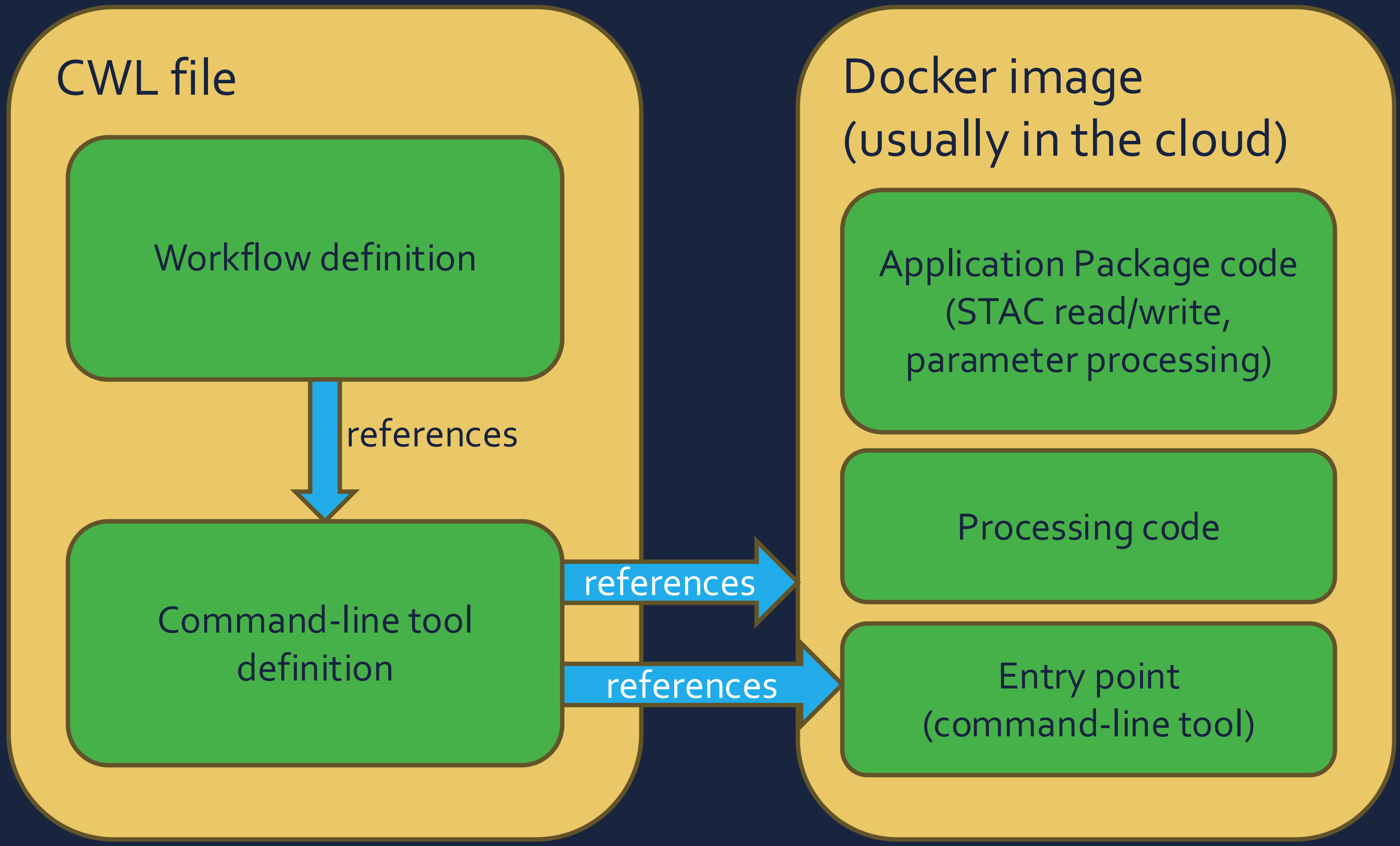
BROCKMANN CONSULT

OGC EO Application Packages

- What's it for?
  - An Application Package provides a **standardized way** to package EO processing software so that it can be run (and combined into workflows) on any compliant platform.

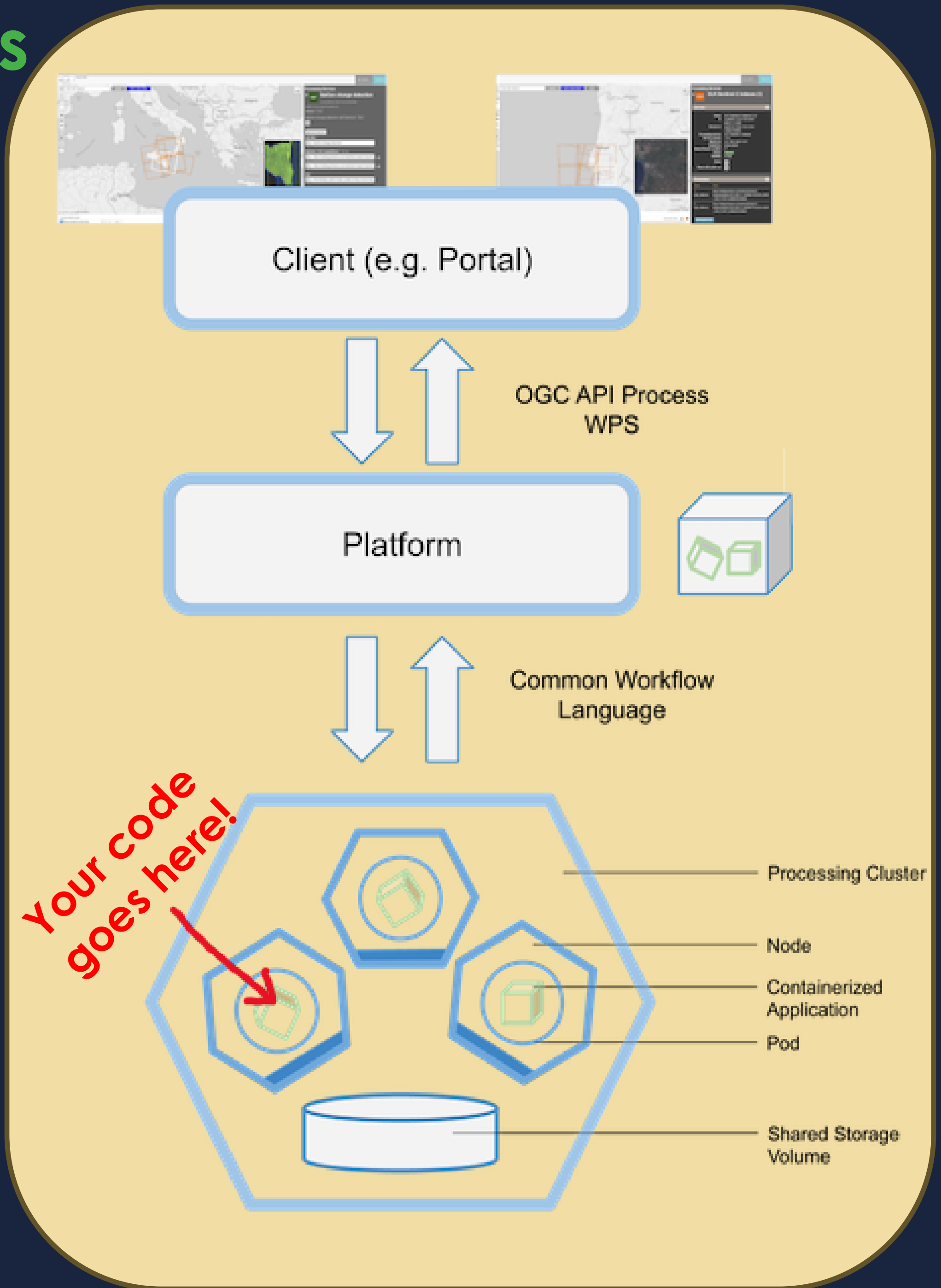
- How does it work? With two components:
  - A **Docker image** which contains the actual code to be run (in any language(s), in any environment).
  - A CWL (**Common Workflow Language**) file which references the Docker image. It details **how to run** the program in the Docker image and what its **inputs and outputs** are.

Structure of an Application Package


CWL file
Workflow definition
references
Command-line tool definition
references
references
Docker image (usually in the cloud)
Application Package code (STAC read/write, parameter processing)
Processing code
Entry point (command-line tool)

Running Application Packages

- An Application Package is designed to be run on an **Application Package Platform**.
- For simple local tests, we can use **cwltool**.
- Note: cwltool runs the CWL, but **doesn't** implement stage-in / stage-out like a real platform.



Client (e.g. Portal)

OGC API Process
WPS

Platform

Common Workflow
Language

Your code goes here!

Processing Cluster

Node

Containerized
Application

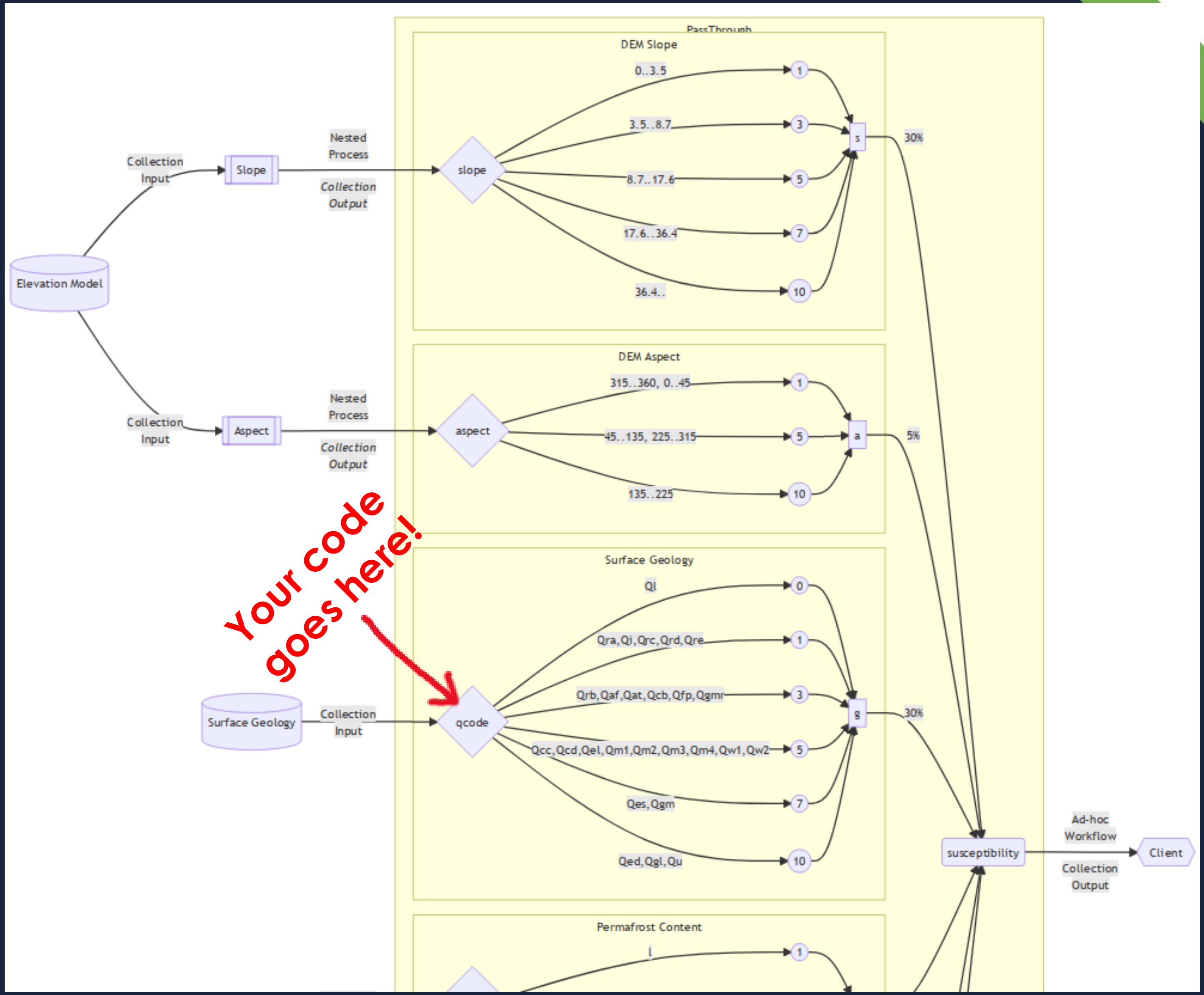
Pod

Shared Storage
Volume

Application Packages in OGC workflows

An Application Package platform lets you chain and combine Application Packages using OGC API – Processes.

Packaged code becomes a FAIR, versatile, reusable building block.

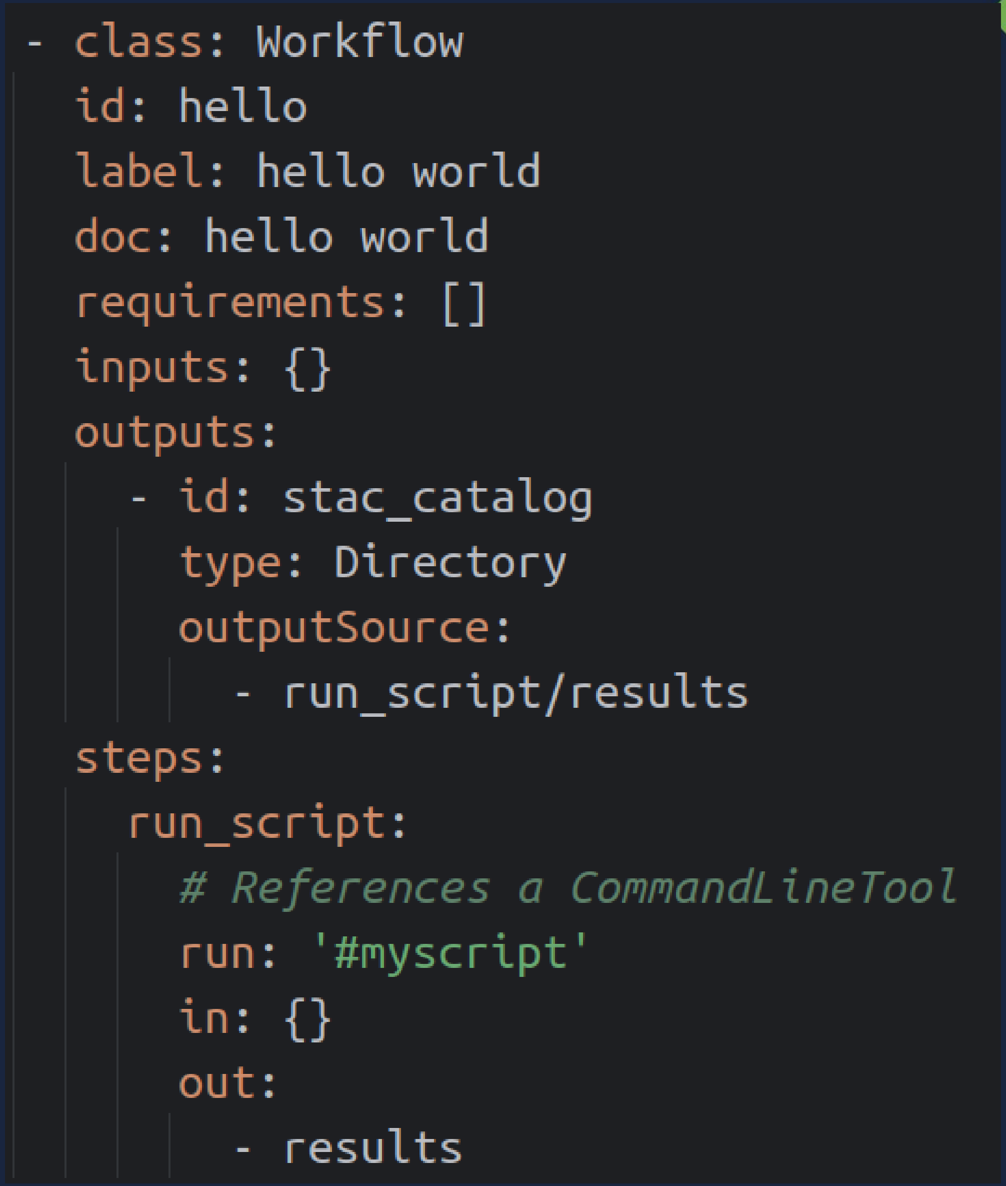


Your code goes here!

What does a CWL file look like?

CWL uses the familiar YAML format, and a simple CWL file has two main parts: a **Workflow** and a **CommandLineTool**.

Here's a simple Workflow with no inputs and one output.



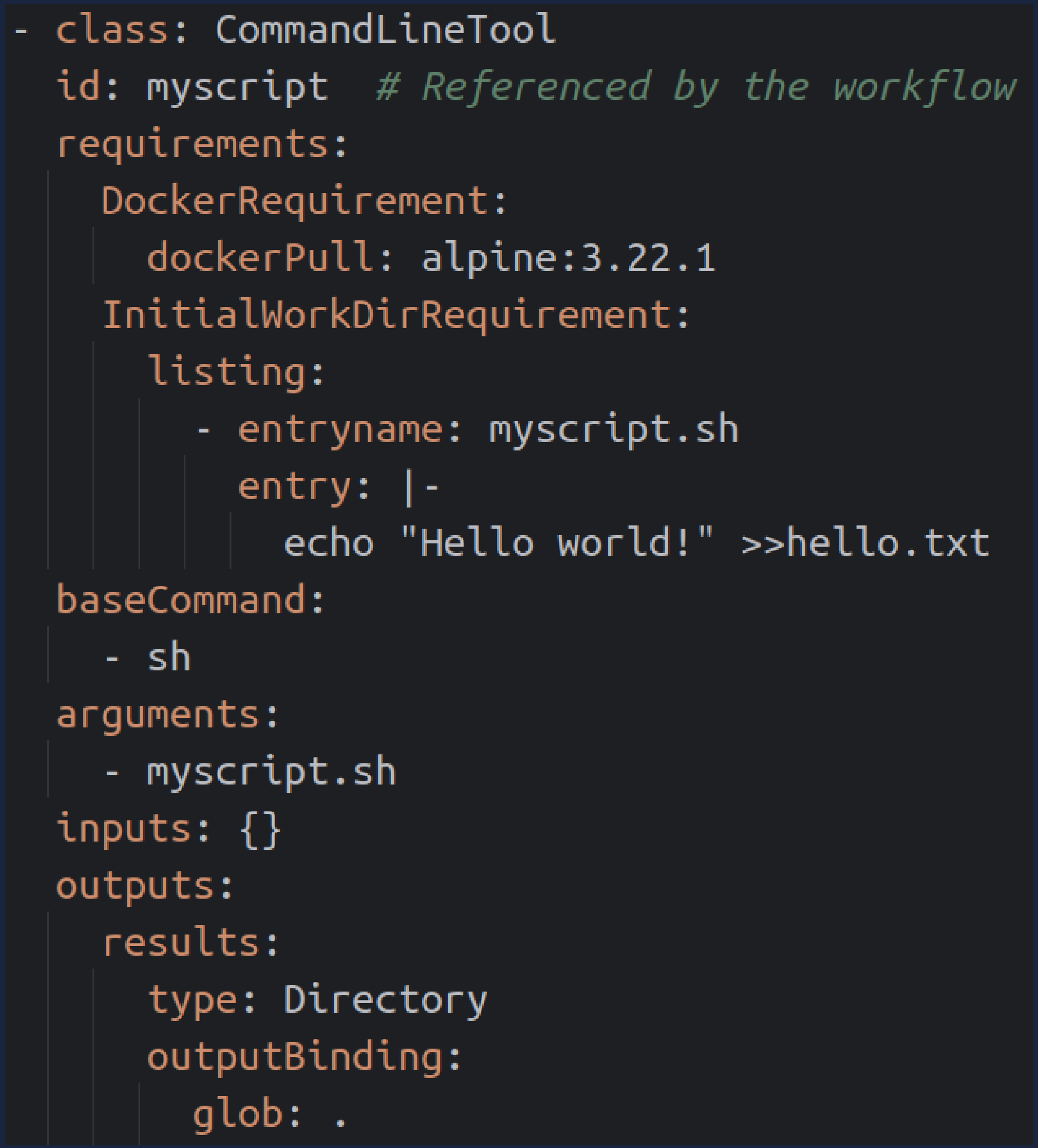
```
- class: Workflow
  id: hello
  label: hello world
  doc: hello world
  requirements: []
  inputs: {}
  outputs:
    - id: stac_catalog
      type: Directory
      outputSource:
        - run_script/results
  steps:
    run_script:
      # References a CommandLineTool
      run: '#myscript'
      in: {}
      out:
        - results
```

What does a CWL file look like? (continued)

The other main part: the CommandLineTool.

- Runs a specified command in a specified Docker image.

- Inputs/outputs are mapped to the Workflow's inputs/outputs.



```
- class: CommandLineTool
  id: myscript  # Referenced by the workflow
  requirements:
    DockerRequirement:
      dockerPull: alpine:3.22.1
    InitialWorkDirRequirement:
      listing:
        - entryname: myscript.sh
          entry: |-
            echo "Hello world!" >>hello.txt
  baseCommand:
    - sh
  arguments:
    - myscript.sh
  inputs: {}
  outputs:
    results:
      type: Directory
      outputBinding:
        glob: .
```

BROCKMANN
CONSULT

Hands on 1:
edit and run an
Application Package

Part 2: From Notebooks to Application Packages with xcengine

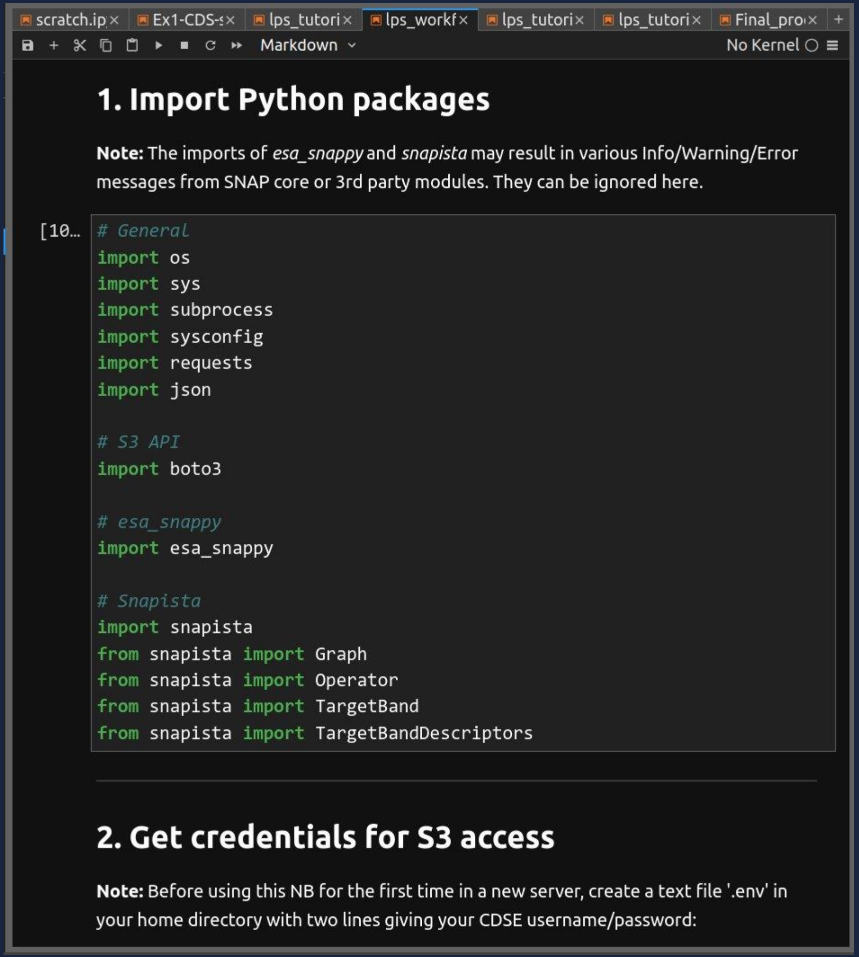
BROCKMANN CONSULT

Comparing Notebooks and Application Packages

Topic	Jupyter Notebook	Application Package
Python environment	Usually handled outside notebook (e.g. conda env file).	Must be set up inside the container.
Parameter handling	Everything's editable, so any variable can be a parameter.	Strictly defined in CWL file, passed to container via CLI.
⚠️ Data input/output	Read and write however and wherever you like.	Stage-in and stage-out via platform using STAC catalogues.
Distribution	The notebook file and something defining the environment.	One CWL file referencing one or more container images.
Paradigm	Primarily interactive.	Strictly batch.

Notebook to Application Package: what's needed?

"Here's my notebook.
Now what do I do?"



1. Import Python packages

Note: The imports of *esa_snappy* and *snapista* may result in various Info/Warning/Error messages from SNAP core or 3rd party modules. They can be ignored here.

[10…

```
# General
import os
import sys
import subprocess
import sysconfig
import requests
import json

# S3 API
import boto3

# esa_snappy
import esa_snappy

# Snapista
import snapista
from snapista import Graph
from snapista import Operator
from snapista import TargetBand
from snapista import TargetBandDescriptors
```

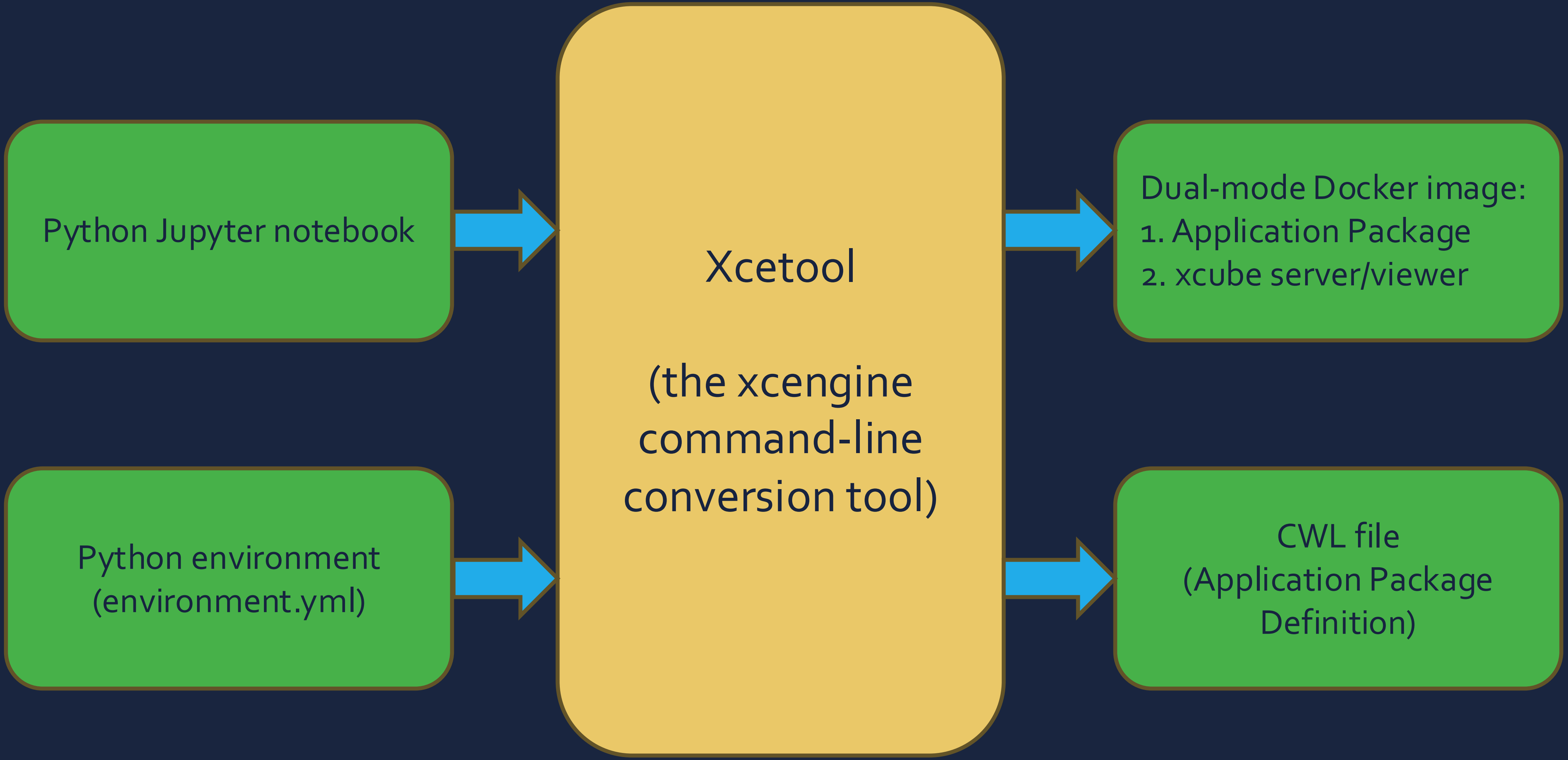
2. Get credentials for S3 access

Note: Before using this NB for the first time in a new server, create a text file '.env' in your home directory with two lines giving your CDSE username/password:

1. Turn it into a runnable script.
2. Process command-line arguments.
3. Read input data via STAC catalogue. (Application Packages **don't** usually fetch their own data!)
4. Write output data via STAC catalogue.
5. Define, build, and publish a container image.
6. Write a CWL file defining inputs/outputs.
7. Test and debug until it works!

Can we automate some of this?

xcengine: turning notebooks into Application Packages



Python Jupyter notebook

Python environment (environment.yml)

Xcetool

(the xcengine command-line conversion tool)

Dual-mode Docker image:
1. Application Package
2. xcube server/viewer

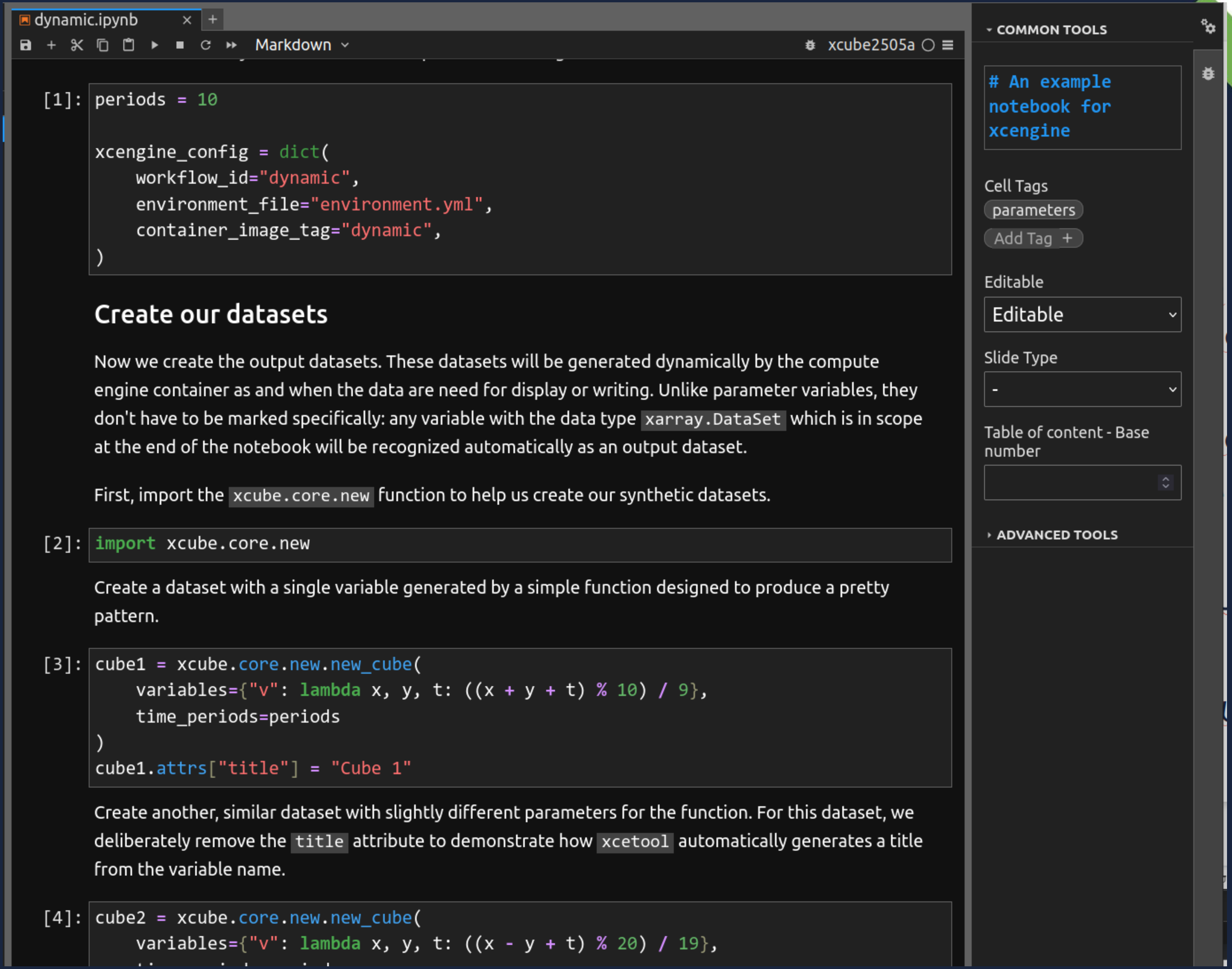
CWL file (Application Package Definition)

An xcengine example

- Parameters
  - There's a tagged "parameters cell" (as also used by Papermill).
  - xcengine automatically turns this into command-line arguments, type definitions, default values, and CWL file boilerplate sections.
- Data stage-out
  - After the code's run, xcengine finds all the datasets and writes them to disk.
  - xcengine generates a valid STAC catalogue for all the written data.
- Environment
  - xcengine sets up an environment in the docker image from a supplied environment file.
  - xcengine can also try to reproduce the current environment.
- The result: an Application Package (container image + CWL file)

An example input notebook for xcengine

In the **xcengine-nb** subdirectory of the repository is the notebook **dynamic.ipynb**, which we'll use for an example.



[1]:
```
periods = 10

xcengine_config = dict(
    workflow_id="dynamic",
    environment_file="environment.yml",
    container_image_tag="dynamic",
)
```

Create our datasets

Now we create the output datasets. These datasets will be generated dynamically by the compute engine container as and when the data are need for display or writing. Unlike parameter variables, they don't have to be marked specifically: any variable with the data type `xarray.DataSet` which is in scope at the end of the notebook will be recognized automatically as an output dataset.

First, import the `xcube.core.new` function to help us create our synthetic datasets.

[2]:
```
import xcube.core.new
```

Create a dataset with a single variable generated by a simple function designed to produce a pretty pattern.

[3]:
```
cube1 = xcube.core.new.new_cube(
    variables={"v": lambda x, y, t: ((x + y + t) % 10) / 9},
    time_periods=periods
)
cube1.attrs["title"] = "Cube 1"
```

Create another, similar dataset with slightly different parameters for the function. For this dataset, we deliberately remove the `title` attribute to demonstrate how `xcetool` automatically generates a title from the variable name.

[4]:
```
cube2 = xcube.core.new.new_cube(
    variables={"v": lambda x, y, t: ((x - y + t) % 20) / 19},
```

```
# An example
notebook for
xcengine
```

Cell Tags
parameters
Add Tag +

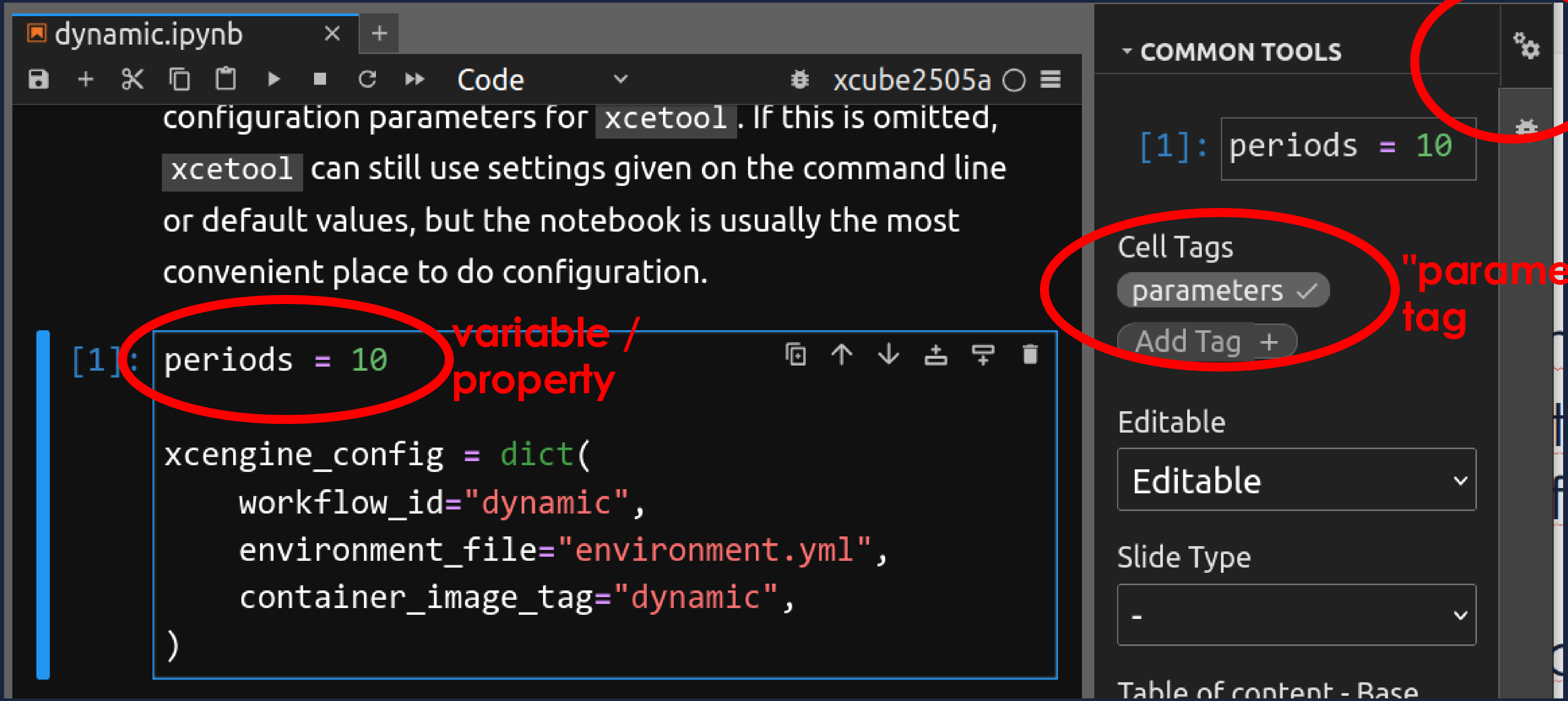
Editable: Editable

Slide Type: -

Table of content - Base number

An example input notebook (continued)

Here's the **parameters cell**, tagged using the Jupyter Lab property inspector. Any variable defined here becomes a parameter.



configuration parameters for `xcetool`. If this is omitted, `xcetool` can still use settings given on the command line or default values, but the notebook is usually the most convenient place to do configuration.

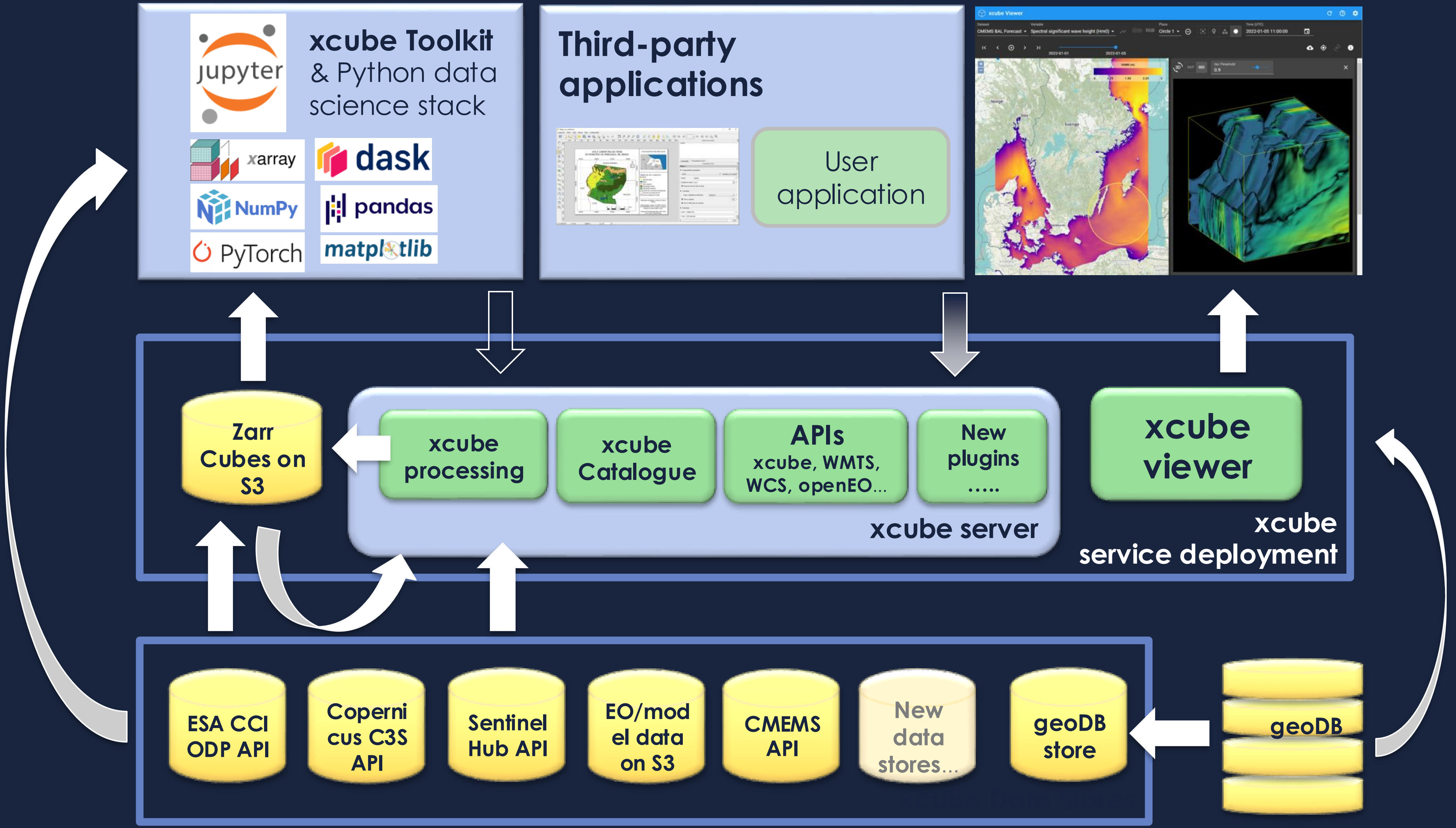
```
[1]: periods = 10

     xcengine_config = dict(
         workflow_id="dynamic",
         environment_file="environment.yml",
         container_image_tag="dynamic",
     )
```

Hands on 2:
use xcengine to create an Application Package

Part 3:
xcube in xcengine Docker images

An overview of the xcube ecosystem



xcube Toolkit & Python data science stack

jupyter

xarray

dask

NumPy

pandas

PyTorch

matplotlib

Third-party applications

User application

Zarr Cubes on S3

xcube processing

xcube Catalogue

APIs xcube, WMTS, WCS, openEO…

New plugins …..

xcube server

xcube viewer

xcube service deployment

ESA CCI ODP API

Copernicus C3S API

Sentinel Hub API

EO/model data on S3

CMEMS API

New data stores…

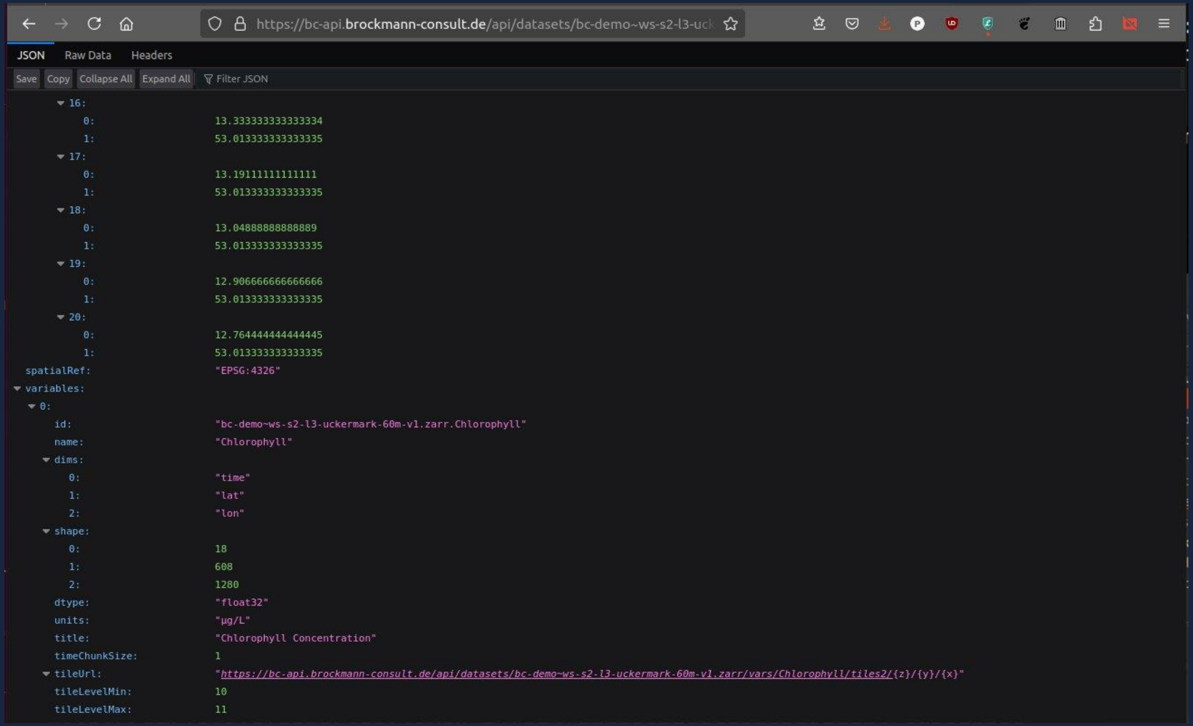
geoDB store

geoDB

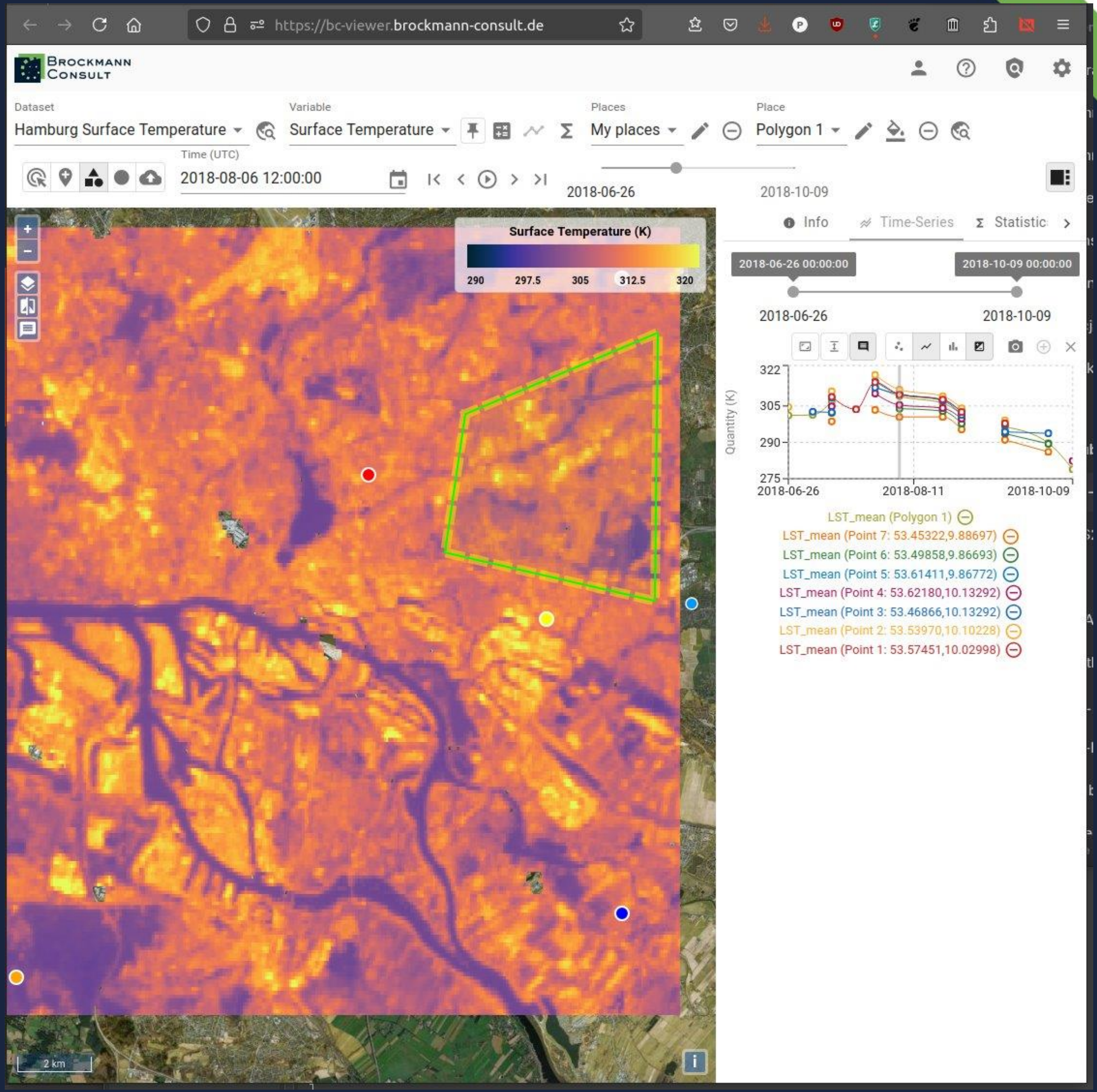
Simplified architecture

xcube server and viewer

These are the parts of xcube we'll be using today: the built-in **API server** and **web viewer**.


https://bc-api.brockmann-consult.de/api/datasets/bc-demo~ws-s2-l3-uck
JSON
Raw Data
Headers
Save
Copy
Collapse All
Expand All
Filter JSON

```
    ▾ 16:
        0:                13.333333333333334
        1:                53.013333333333335
    ▾ 17:
        0:                13.19111111111111
        1:                53.013333333333335
    ▾ 18:
        0:                13.04888888888889
        1:                53.013333333333335
    ▾ 19:
        0:                12.906666666666666
        1:                53.013333333333335
    ▾ 20:
        0:                12.764444444444445
        1:                53.013333333333335
  spatialRef:             "EPSG:4326"
▾ variables:
  ▾ 0:
      id:                 "bc-demo~ws-s2-l3-uckermark-60m-v1.zarr.Chlorophyll"
      name:               "Chlorophyll"
    ▾ dims:
        0:                "time"
        1:                "lat"
        2:                "lon"
    ▾ shape:
        0:                18
        1:                608
        2:                1280
      dtype:              "float32"
      units:              "µg/L"
      title:              "Chlorophyll Concentration"
      timeChunkSize:      1
    ▾ tileUrl:            "https://bc-api.brockmann-consult.de/api/datasets/bc-demo~ws-s2-l3-uckermark-60m-v1.zarr/vars/Chlorophyll/tiles2/{z}/{y}/{x}"
      tileLevelMin:       10
      tileLevelMax:       11
```



Dataset: Hamburg Surface Temperature

Variable: Surface Temperature

Places: My places

Place: Polygon 1

Time (UTC): 2018-08-06 12:00:00

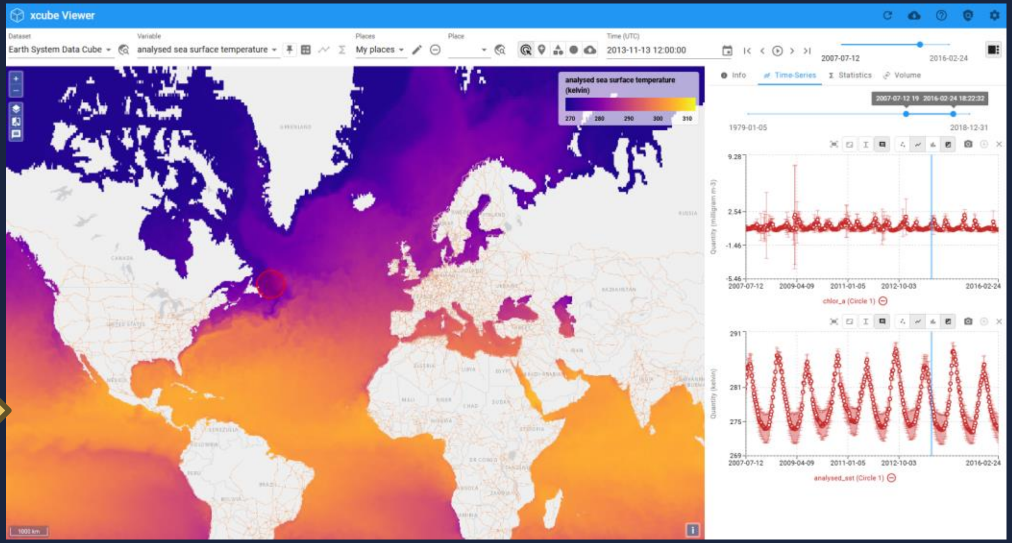
2018-06-26 00:00:00 – 2018-10-09 00:00:00

LST_mean (Polygon 1)
LST_mean (Point 7: 53.45322,9.88697)
LST_mean (Point 6: 53.49858,9.86693)
LST_mean (Point 5: 53.61411,9.86772)
LST_mean (Point 4: 53.62180,10.13292)
LST_mean (Point 3: 53.46866,10.13292)
LST_mean (Point 2: 53.53970,10.10228)
LST_mean (Point 1: 53.57451,10.02998)

The xcengine Docker image includes an xcube server

- Run the container image through the CWL file, and it's an Application Package.
- Run it with the "--server" parameter, and it's "xcube in a box".
- An easy way to distribute an xcube viewer and API server preconfigured to show/serve your data.

```
xcetool image run --server dynamic:1
```



Dataset
Earth System Data Cube

Variable
analysed sea surface temperature

Places
My places

Time (UTC)
2013-11-13 12:00:00

Hands on 3:
run the Docker image as an xcube server

BROCKMANN
CONSULT

Part 4 (optional): Deploying an Application Package platform

Setting up an Application Package platform

- This is optional material for the course – we're mainly concerned with *making* Application Packages here.

- But setting up an Application Package platform gives a useful view of the broader ecosystem.

- And of course it can be useful for testing and debugging.

Introducing the ZOO-Project

- ZOO (https://zoo-project.org/) is a mature, open-source WPS (Web Processing Service) platform.

- More recently, support has been implemented for OGC API – Processes Part 1 & 2, including support for Application Packages as processors.

- It's a relatively heavyweight, cluster-based system, but can be run on Minikube on a personal computer.

Brief ZOO set-up guide

Install helm, minikube, and kubectl, then:

```
minikube start
helm upgrade -i zoo-project-dru \
  zoo-project/zoo-project-dru \
  --namespace zoo --create-namespace \
  --values https://raw.githubusercontent.com/ZOO-Project/charts/refs/heads/main/zoo-project-dru/values_minikube.yaml
```

Then follow helm's instructions to set up port forwarding.