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
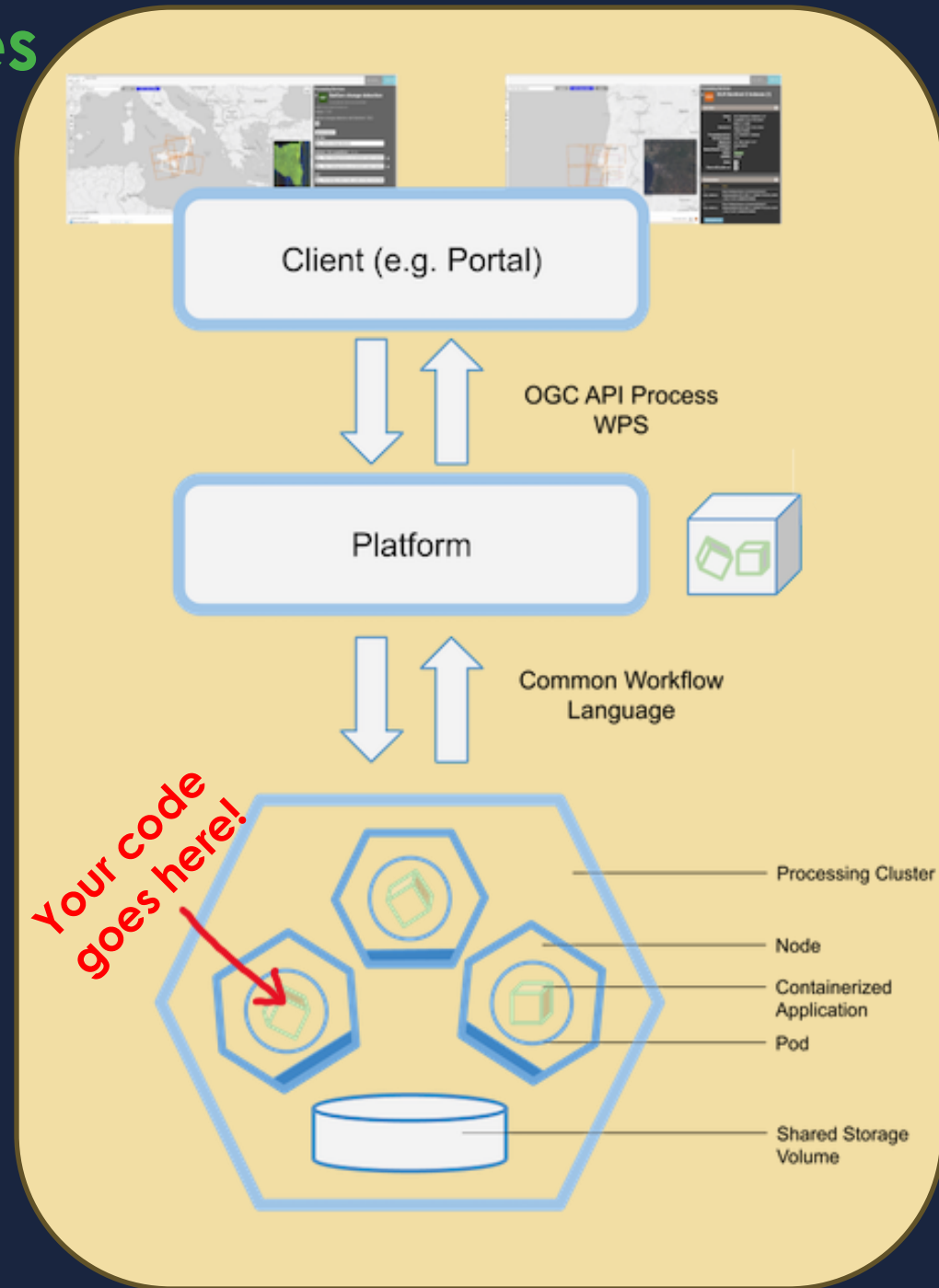# Application Packages

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
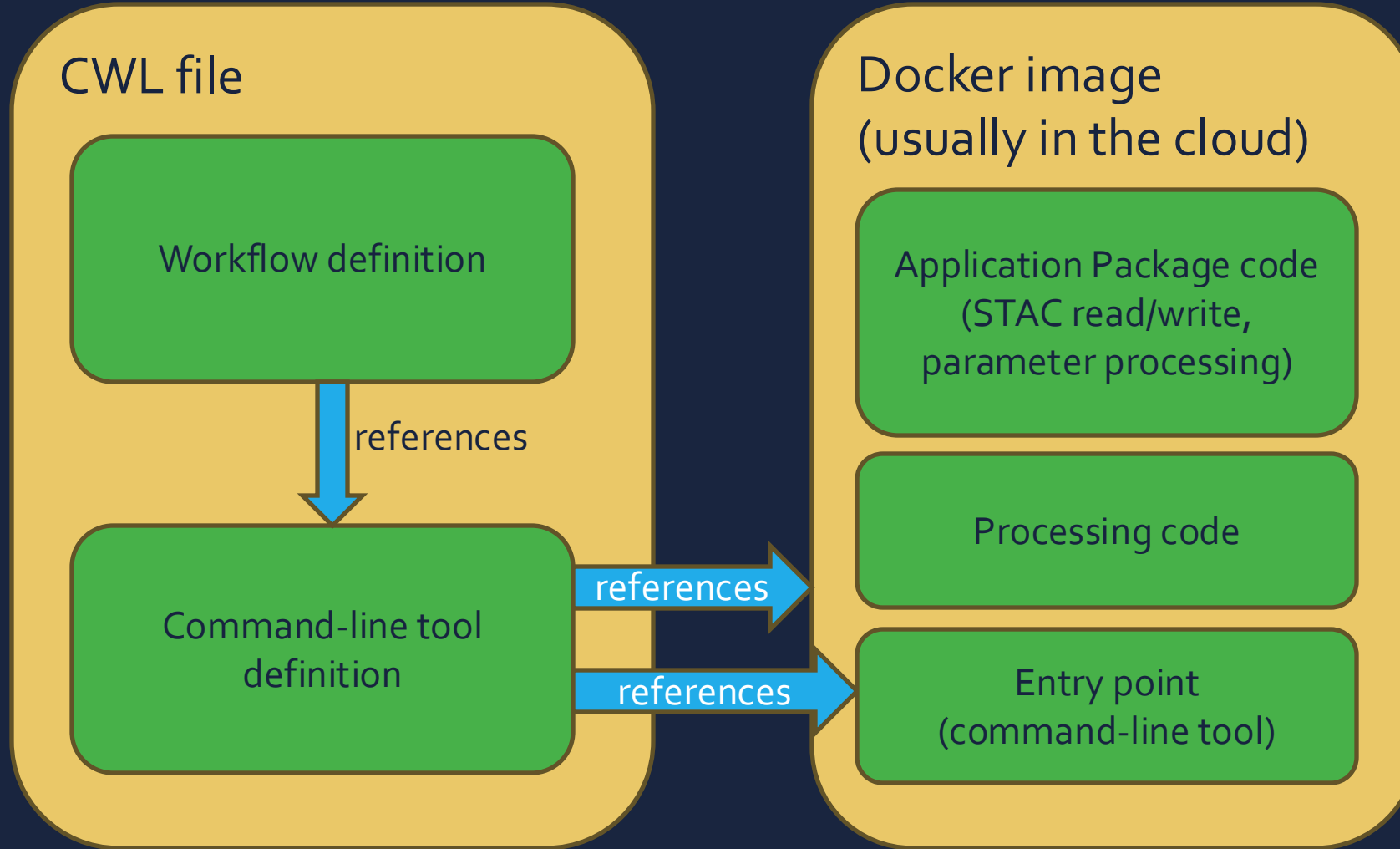
Running Application Packages

- An Application Package is designed to be run on an **Application Package Platform**.

- For simple local tests, we can use **cwltool**.

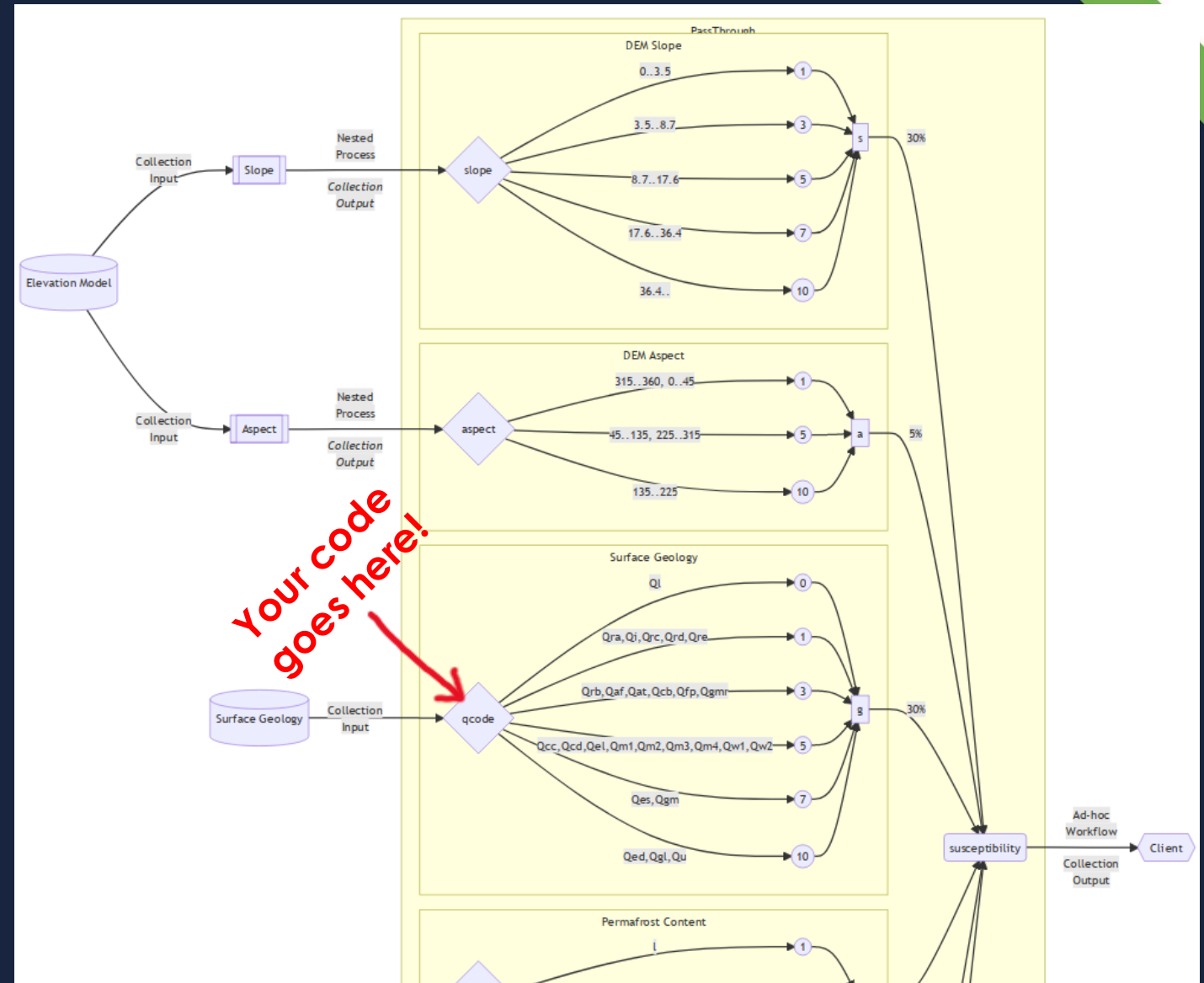- Note: cwltool runs the CWL, but **doesn't** implement stage-in / stage-out like a real platform.

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.

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
```

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: .
```

Hands on 1:
edit and run an
Application Package

BROCKMANN
CONSULT
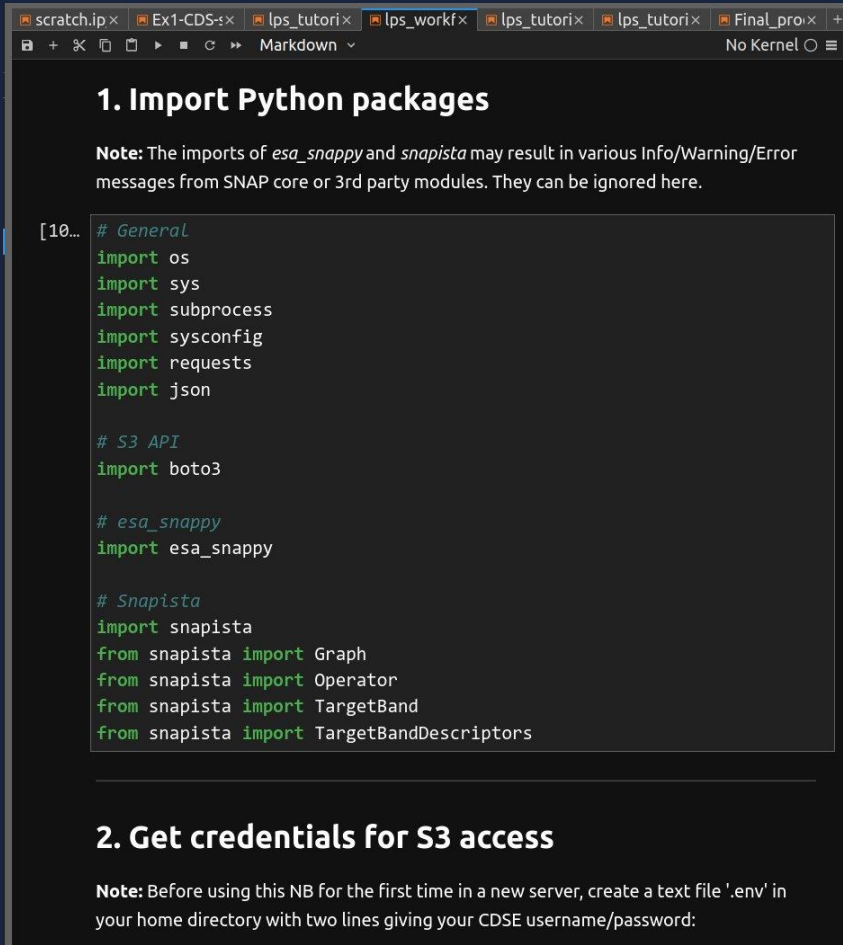
Part 2:
From Notebooks to Application Packages with xcengine

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. 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!
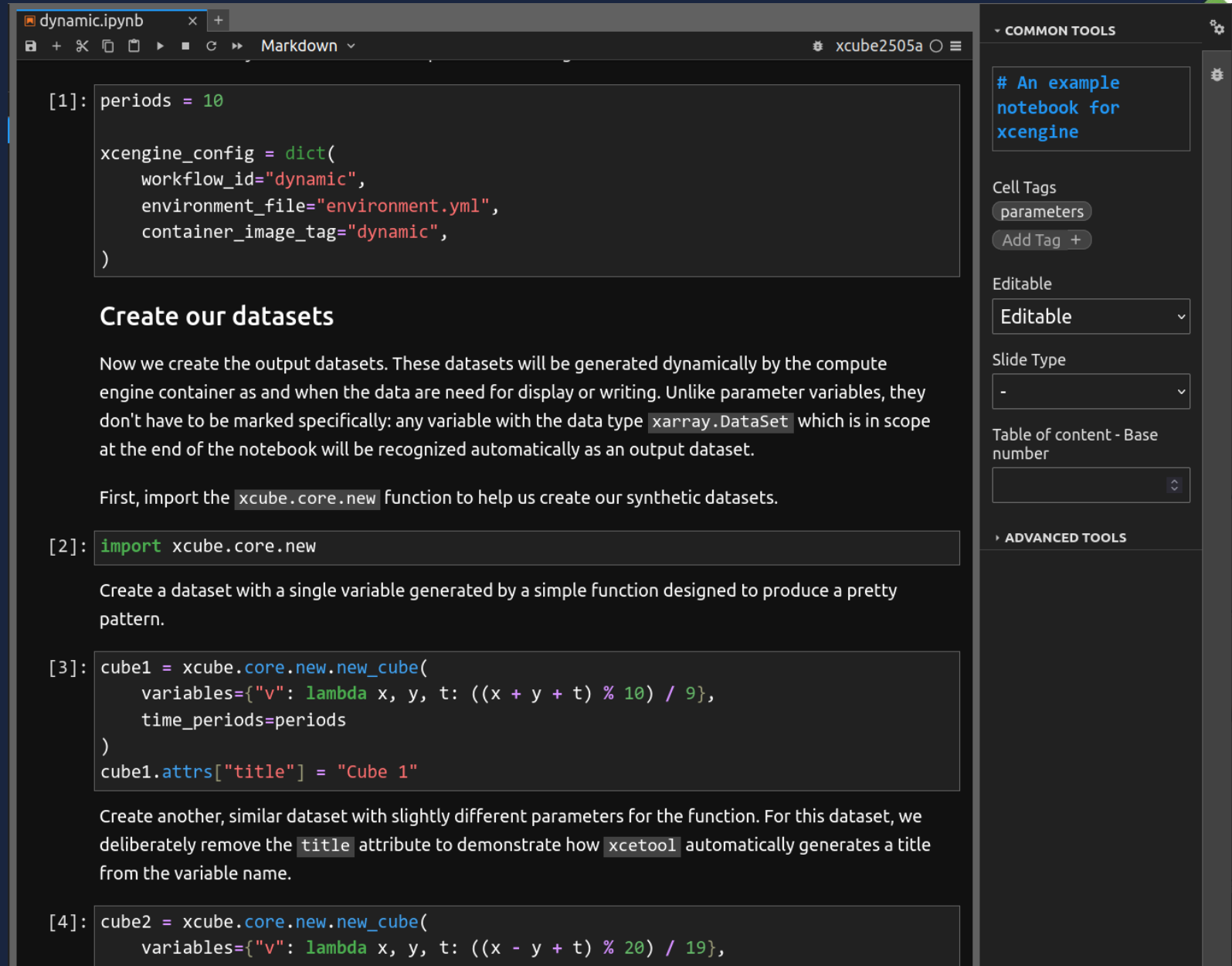
Can we automate some of this?

xcengine: turning notebooks into Application Packages

Python Jupyter notebook → Xcetool (the xcengine command-line conversion tool) → Dual-mode Docker image:
1. Application Package
2. xcube server/viewer

Python environment (environment.yml) → Xcetool (the xcengine command-line conversion tool) → CWL file (Application Package Definition)
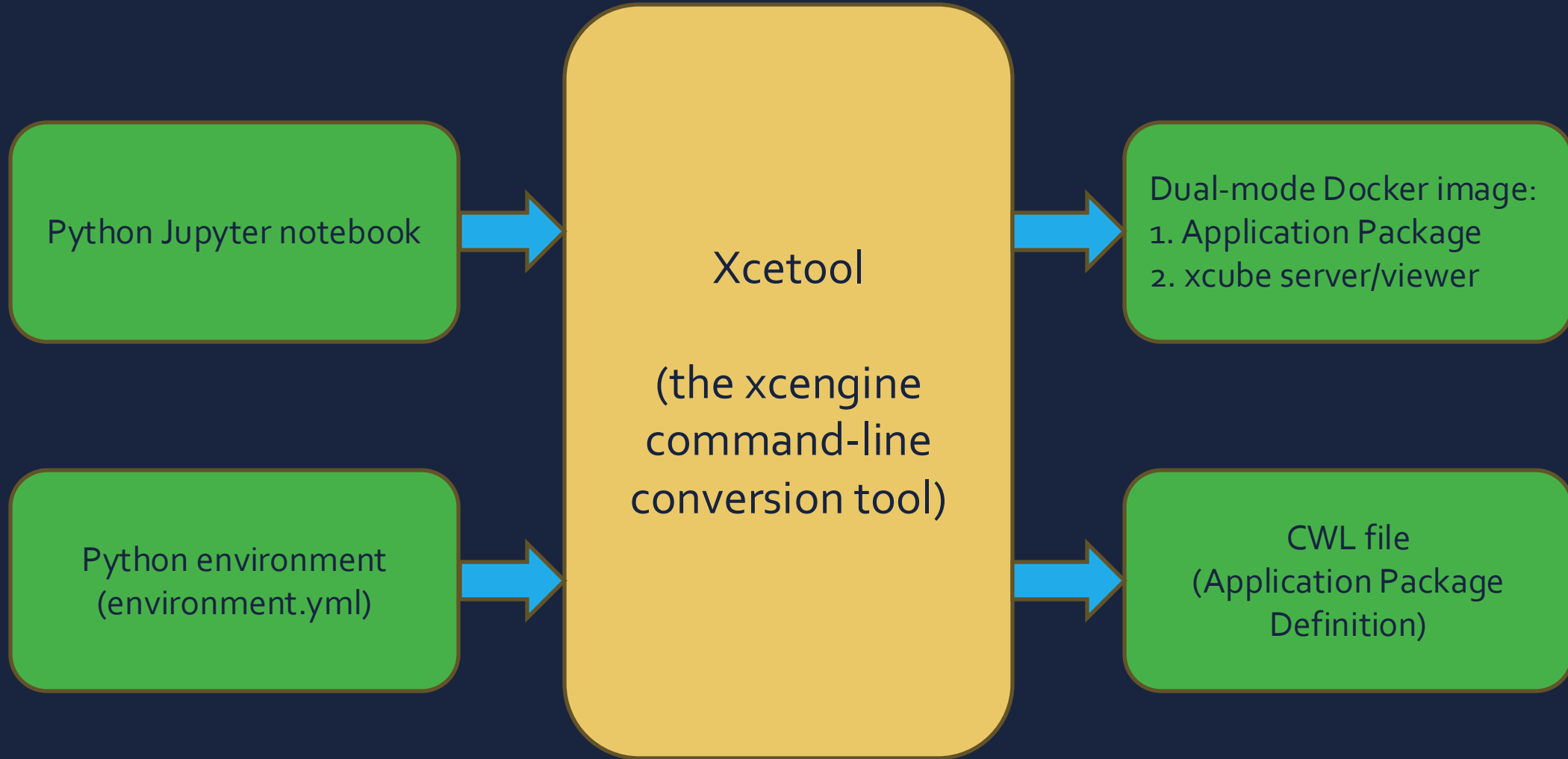
Hands on 2:
use xcengine to create
an Application Package

The input notebook

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

The input notebook (continued)

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

Part 3:
xcube in xcengine Docker images

xcube server and viewer

These are the parts of xcube we'll be using today: the built-in **API server** and **web viewer**.
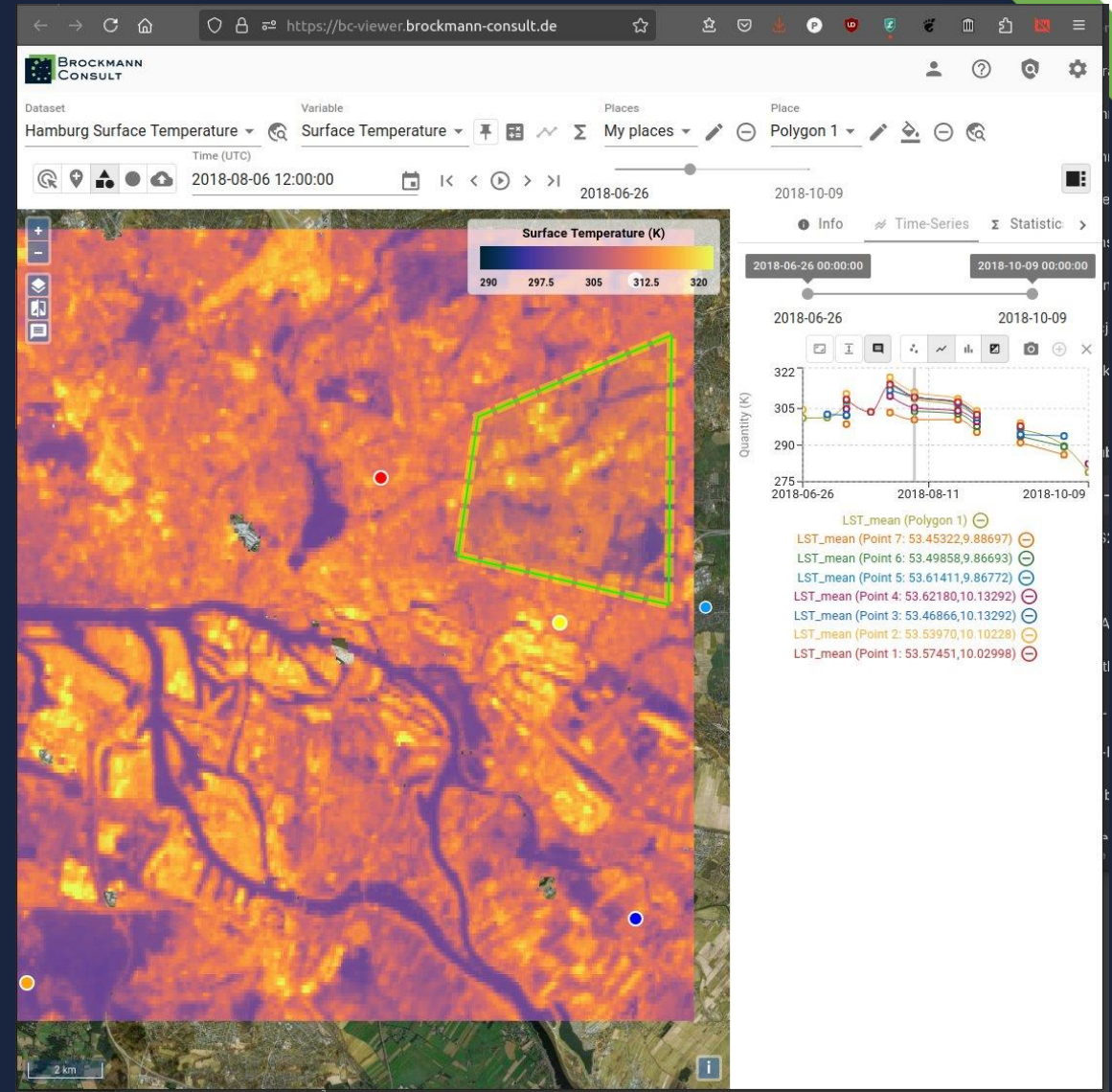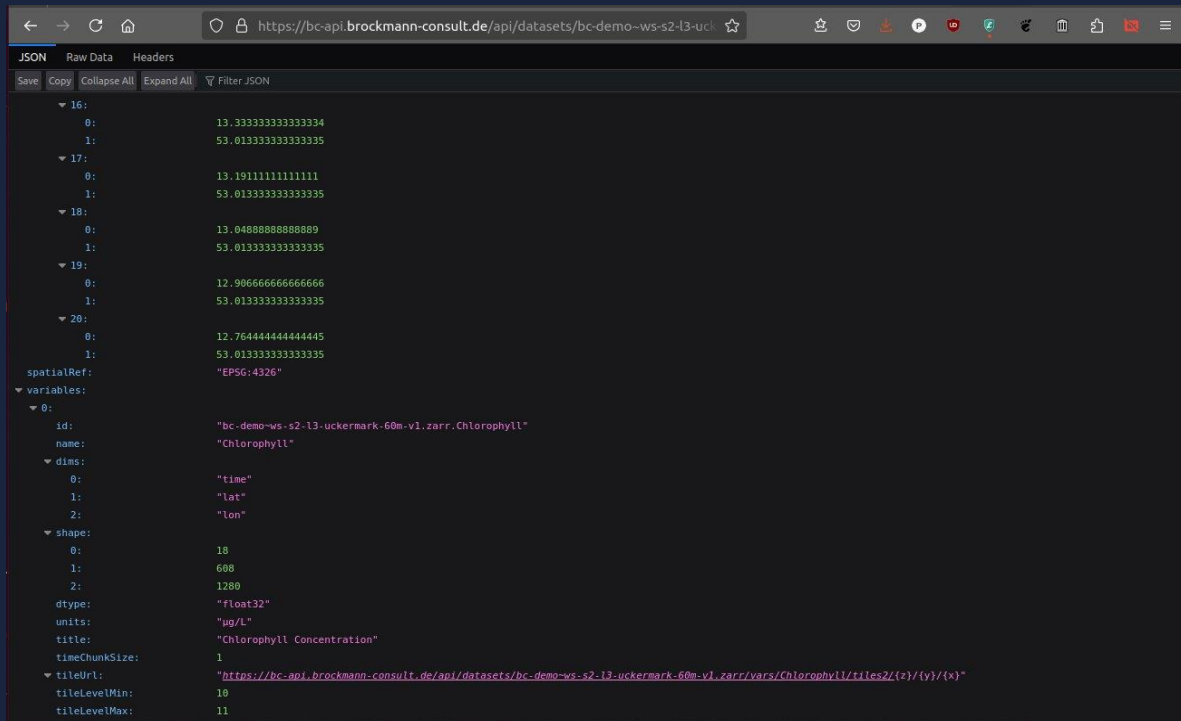
Hands on 3: running the xcengine Docker image in "xcube mode"