

# Remote Sensing -Deployable Analysis Environment

Power your workflow with HPC

netherlands  
eScience center

**SURF**

Meiert W. Grootes, Utrecht 23-01-2024

# The RS-DAT team

NLeSC:

Team Atlas (Pranav Chandramouli, Meiert Grootes, Ou Ku, Francesco Nattino, Fakareh (Sarah) Alidoost, Yifat Dzigan)

SURF:

Ander Astudillo, Martin Brandt, Natalie Danezi, Robert Griffioen, Annette Langedijk, Raymond Oonk

netherlands  
eScience center

**SURF**

“Empowering researchers  
across all disciplines  
through advanced  
research software”

# The Netherlands eScience Center

National centre /  
independent foundation  
since 2012 /  
NWO & SURF





**Let's stay  
in touch**

**Check for  
our open  
calls**

 [www.eScienceCenter.nl](http://www.eScienceCenter.nl)

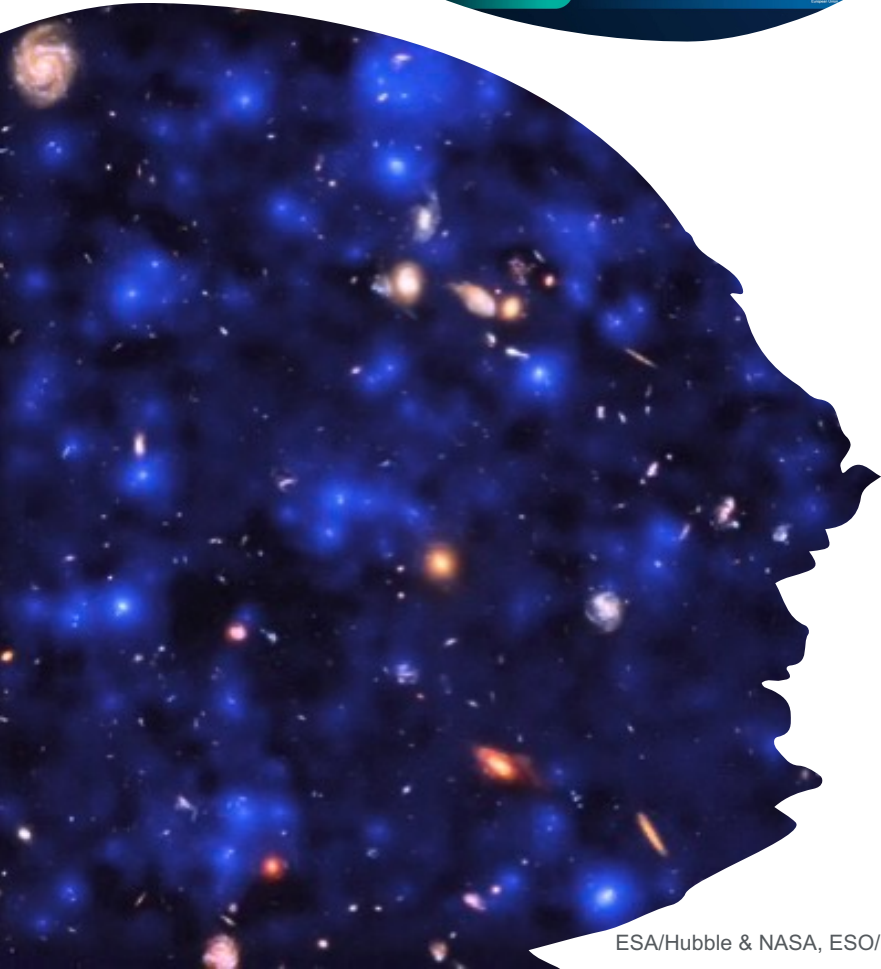
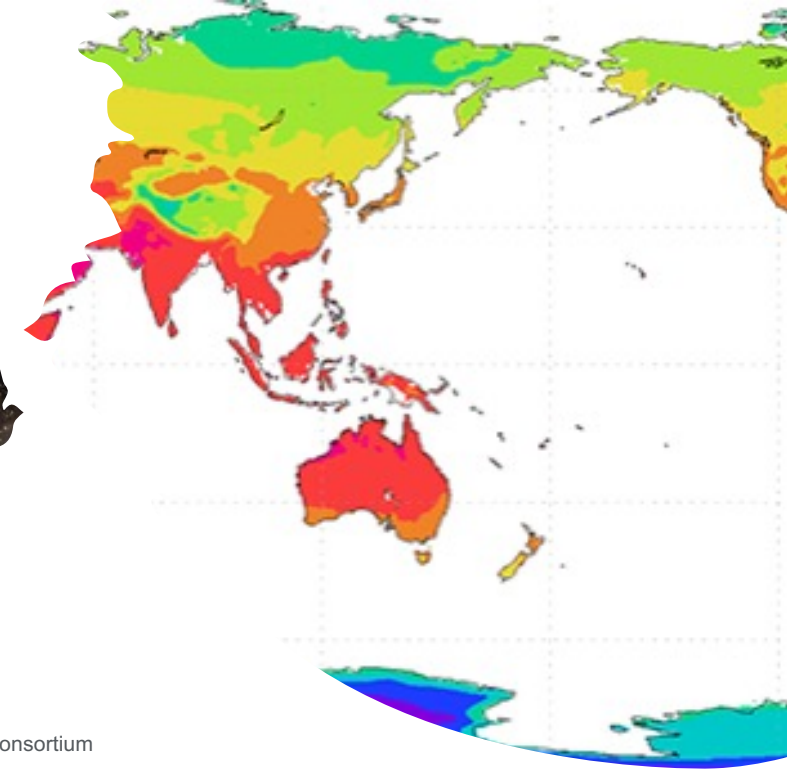
 [info@esciencecenter.nl](mailto:info@esciencecenter.nl)

 +31 20 460 4770

e

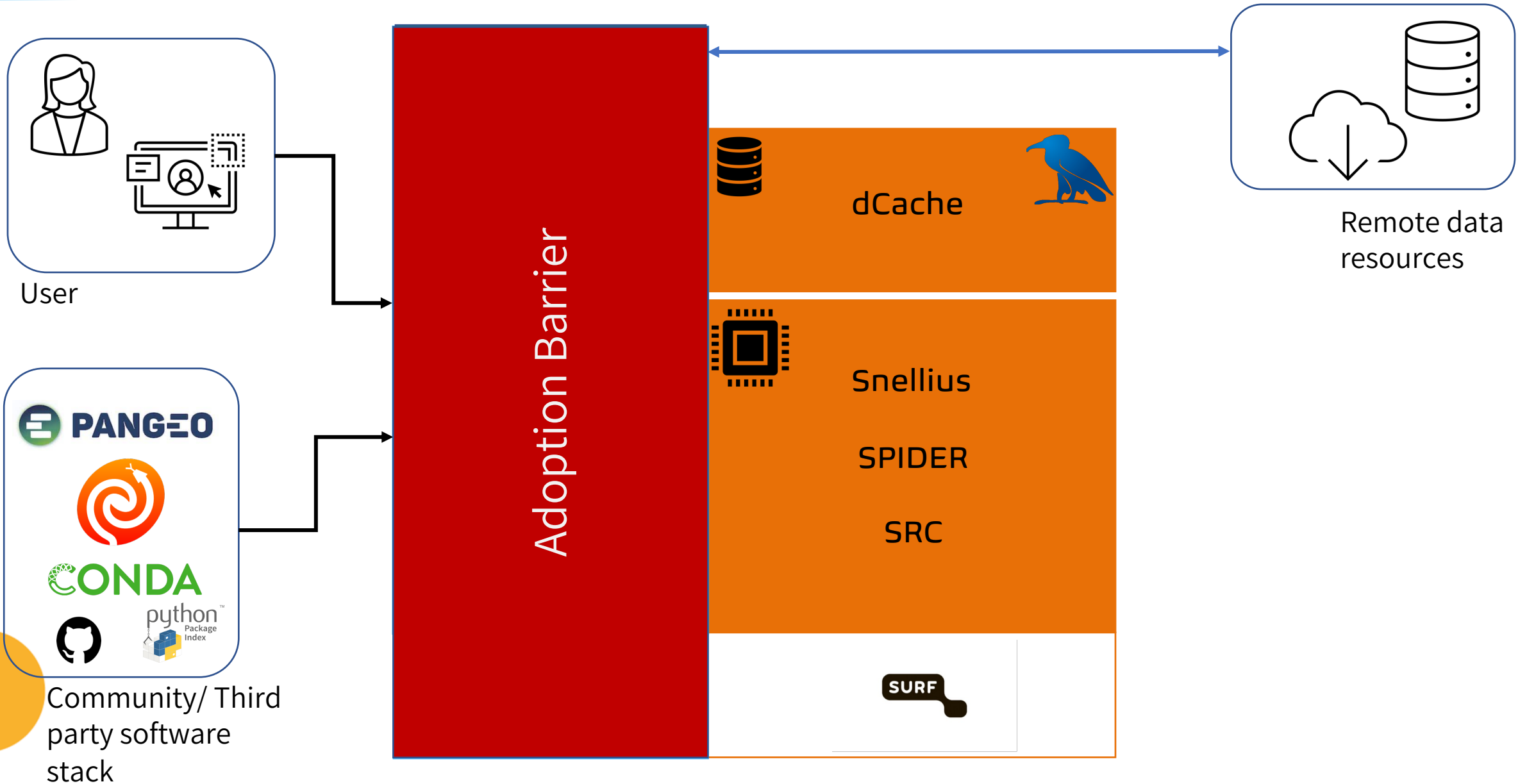


ESA Euclid consortium

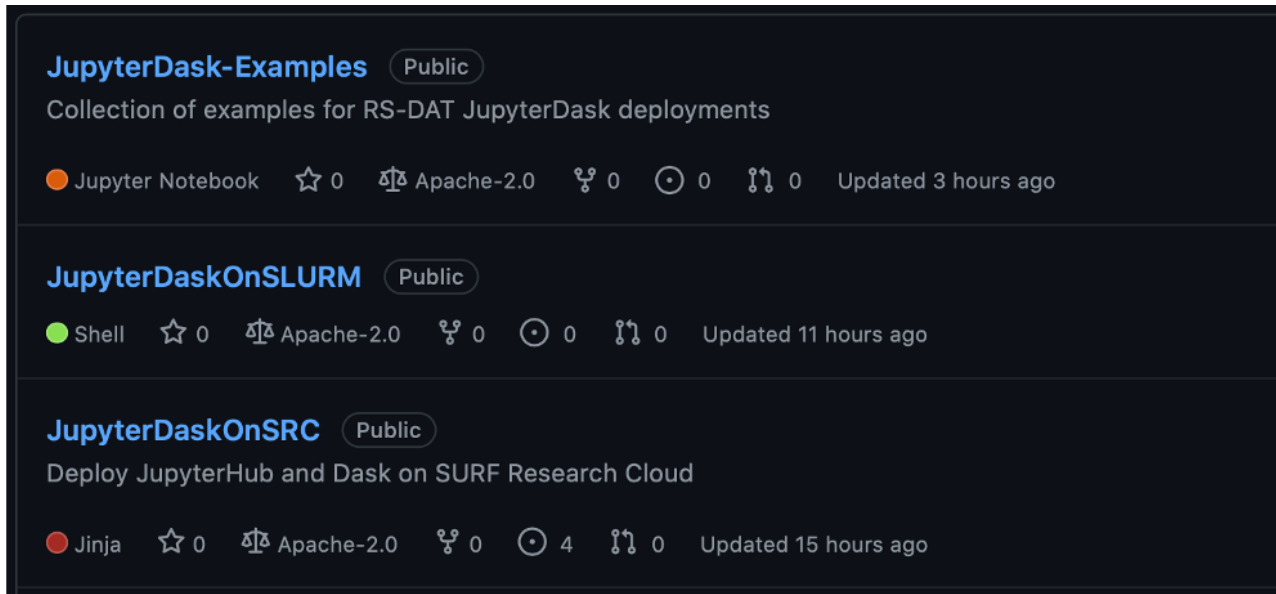


# Democratizing Big Data

- Big Data ubiquitous across EO and astronomy (EO/astro sats, imaging surveys, large FOV IFUs, sims)
- Fundamental challenge – storage & processing
- Low barrier (local) processing solutions do not scale
- Big data solutions often limited by upfront investment (time/money) and steep learning curve



<https://github.com/RS-DAT/JupyterDaskOnSLURM>

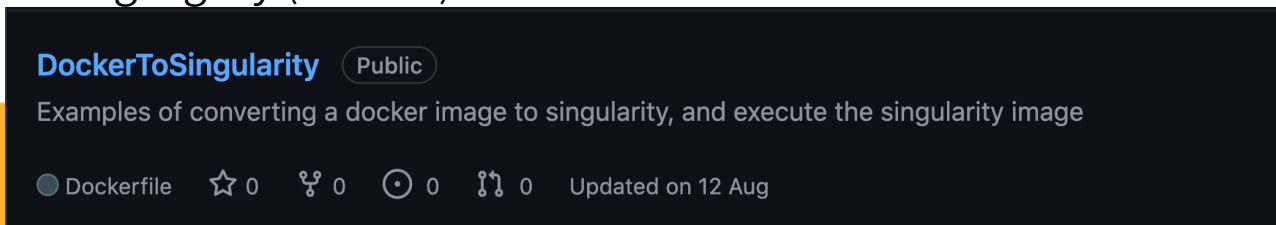


The screenshot shows three GitHub repository listings on a dark background. Each listing includes the repository name, a 'Public' badge, a description, and a row of icons for file type, stars, license, forks, issues, and pull requests, along with the last update time.

- JupyterDask-Examples** (Public): Collection of examples for RS-DAT JupyterDask deployments. Updated 3 hours ago.
- JupyterDaskOnSLURM** (Public): Updated 11 hours ago.
- JupyterDaskOnSRC** (Public): Deploy JupyterHub and Dask on SURF Research Cloud. Updated 15 hours ago.

Scalable analysis with Jupyter

Using legacy (Docker) containers for HPC

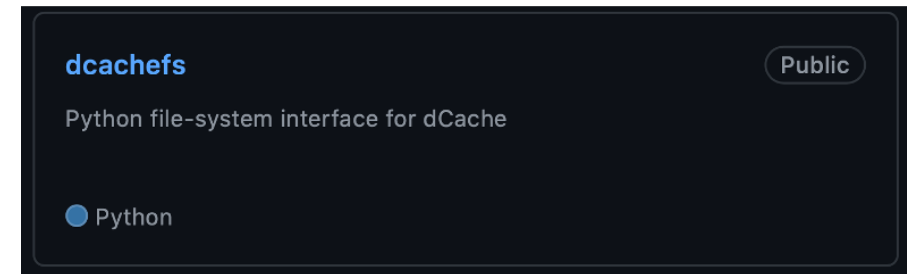


The screenshot shows the GitHub repository listing for DockerToSingularity. It includes the repository name, a 'Public' badge, a description, and a row of icons for file type, stars, forks, issues, and pull requests, along with the last update time.

- DockerToSingularity** (Public): Examples of converting a docker image to singularity, and execute the singularity image. Updated on 12 Aug.

<https://github.com/RS-DAT/DockerToSingularity>

<https://github.com/NLeSC-GO-common-infrastructure/dcachefs>

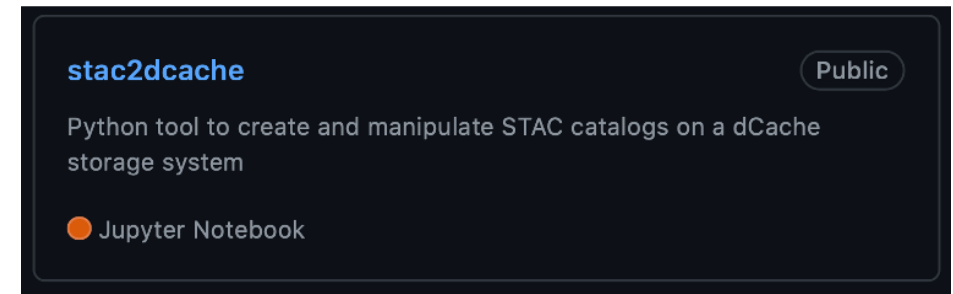


The screenshot shows the GitHub repository listing for dcachefs. It includes the repository name, a 'Public' badge, a description, and a row of icons for file type, stars, forks, issues, and pull requests, along with the last update time.

- dcachefs** (Public): Python file-system interface for dCache. Updated 11 hours ago.

Python interface to dCache

<https://github.com/NLeSC-GO-common-infrastructure/stac2dcache>



The screenshot shows the GitHub repository listing for stac2dcache. It includes the repository name, a 'Public' badge, a description, and a row of icons for file type, stars, forks, issues, and pull requests, along with the last update time.

- stac2dcache** (Public): Python tool to create and manipulate STAC catalogs on a dCache storage system. Updated 11 hours ago.

utility functions to manage STAC catalogs (and the underlying data) on dCache.



# Data storage and access - dCache

## dCache

- Mass storage system supporting heterogenous nodes as single virtual filesystem
- Multiple I/O, incl. HTTP/Webdav
- Powerful system but not fully integrated with (python) analysis workflows



- Python-based, extending Filesystem Spec (`fsspec`).
- Mainly, one filesystem class following `fsspec`'s API (implementing functions like `ls`, `rm`, `cat`, etc.).
- File like objects to read with other libraries
- Automatic integration with other libraries using `fsspec`

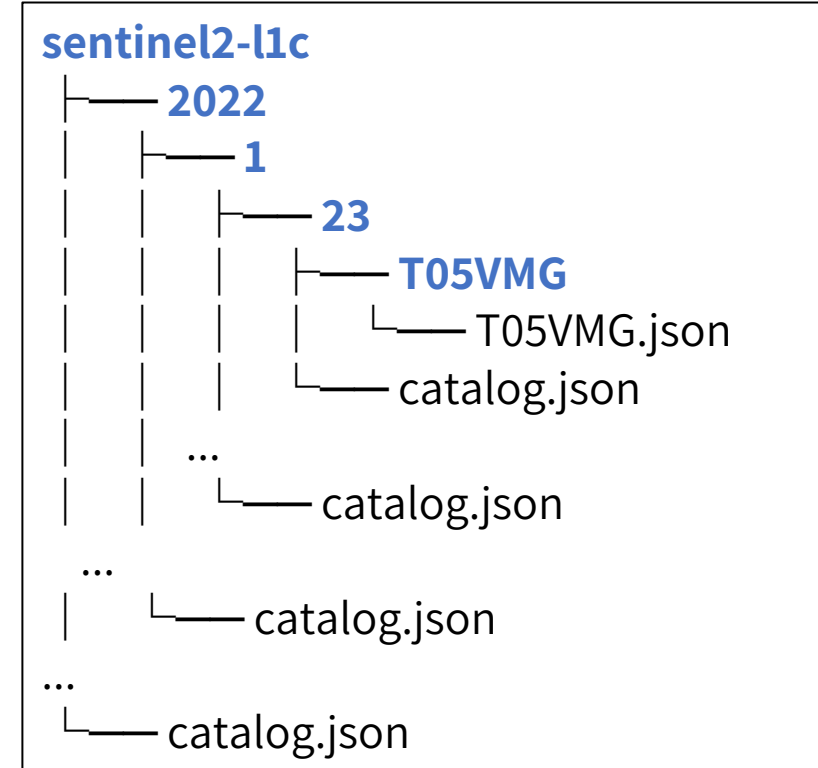
```
import dcachefs
import fsspec

with fsspec.open('https://url/to/myfile.tif', 'r') as fr:
    with fsspec.open('dcache://path/to/myfile.tif', 'w') as fw:
        fw.write(fr.read())
```



## Spatio-Temporal Asset Catalogue

- Common structure to describe spatio-temporal data.
- Standard way to catalog geospatial data files.
- All about metadata, linked to data.
- STAC ecosystem includes:
  - Specifications (core elements definitions);
  - API;
  - Tools.



- STAC2dCache extends the I/O functionality of core STAC library (PySTAC) to read/write metadata from/to the dCache storage.
- Provides utility functions to copy data files to/from dCache

```
import stac2dcache
from pystac import Catalog

catalog = Catalog(...)
catalog.add_items(...)
catalog.normalize_and_save(
    'dcache://path/to/catalog'
)
```

```
from stac2dcache.utils import copy_asset, get_asset

catalog = Catalog.from_file('dcache://path/to/catalog.json')

for asset_key in ('red', 'green', 'blue'):
    copy_asset(catalog, asset_key, update_catalog=True)

da = get_asset(catalog, 'blue', 'T05VMG')
da.plot()
```



# A (Big) Data analysis ecosystem

Pangeo is first and foremost a **community of people** working collaboratively to develop software and infrastructure to enable Big Data geoscience research.

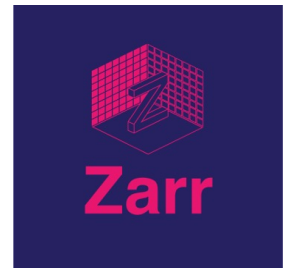
Some of the products produced by this community include interconnected **software package** and **deployments** of this software in cloud and high-performance-computing environments. Such a deployment is sometimes referred to as a *Pangeo Environment*.

The Astropy Project is a community effort to develop a **common core package** for Astronomy in Python and foster an ecosystem of **interoperable astronomy packages**.



PANGEO

A community platform for Big Data geoscience



Scalable computation

Rich data model w/ out-of-core support

Interactive analysis and execution

Flexible storage

# Scalable analysis - Jupyter at scale

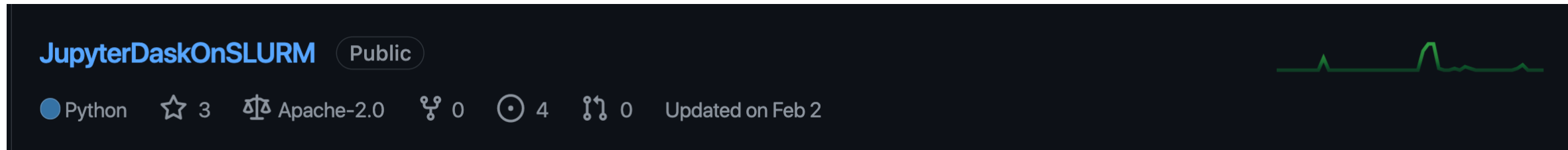
The screenshot displays the JupyterLab web interface in a browser window. The address bar shows 'localhost:8889/lab/tree/Untitled.ipynb'. The interface is divided into several sections:

- Left Panel (File Browser):** Contains a message: "You are not currently in a Git repository. To use Git, navigate to a local repository, initialize a repository here, or clone an existing repository." Below this are three buttons: "Open the FileBrowser", "Initialize a Repository", and "Clone a Repository".
- Center Panel (Code Editor):** Shows an open file named "Untitled.ipynb" with a code editor area containing a single line of code: "[ ]:".
- Right Panel (System Monitoring):** Displays a dashboard with a list of system metrics, each in a yellow bar: MEMORY BY KEY, NPROCESSING, OCCUPANCY, PROFILE, PROFILE SERVER, PROGRESS, SCHEDULER SYSTEM, TASK STREAM, WORKERS, WORKERS CPU TIMESERIES, WORKERS DISK, WORKERS DISK TIMESERIES, WORKERS MEMORY, WORKERS MEMORY TIMESERIES, WORKERS NETWORK, WORKERS NETWORK TIMESERIES, and WORKERS TRANSFER BYTES. Below this is a "CLUSTERS" section with a "+ NEW" button and a "SLURMCluster" entry showing details like Scheduler Address, Dashboard URL, Number of Cores, Memory, and Number of Workers. At the bottom of the cluster entry are "SCALE" and "SHUTDOWN" buttons.
- Bottom Panel (Kernel Status):** Shows "Python 3 (ipykernel) | Idle" with a "Simple" toggle and a "0" indicator.

- Scale analysis: accustomed interactive workflows, but backed by HPC/Cloud
- Support for PyData/Python ecosystem
- Combine Jupyter server with Dask cluster on HPC/HTC/Cluster/Cloud system
- Fully integrated with storage (dCache)
- Expose Jupyter python ecosystem (lab plugins ...)



SURF H\*C Infrastructure: Spider (HTC), Snellius (HPC); SLURM scheduler

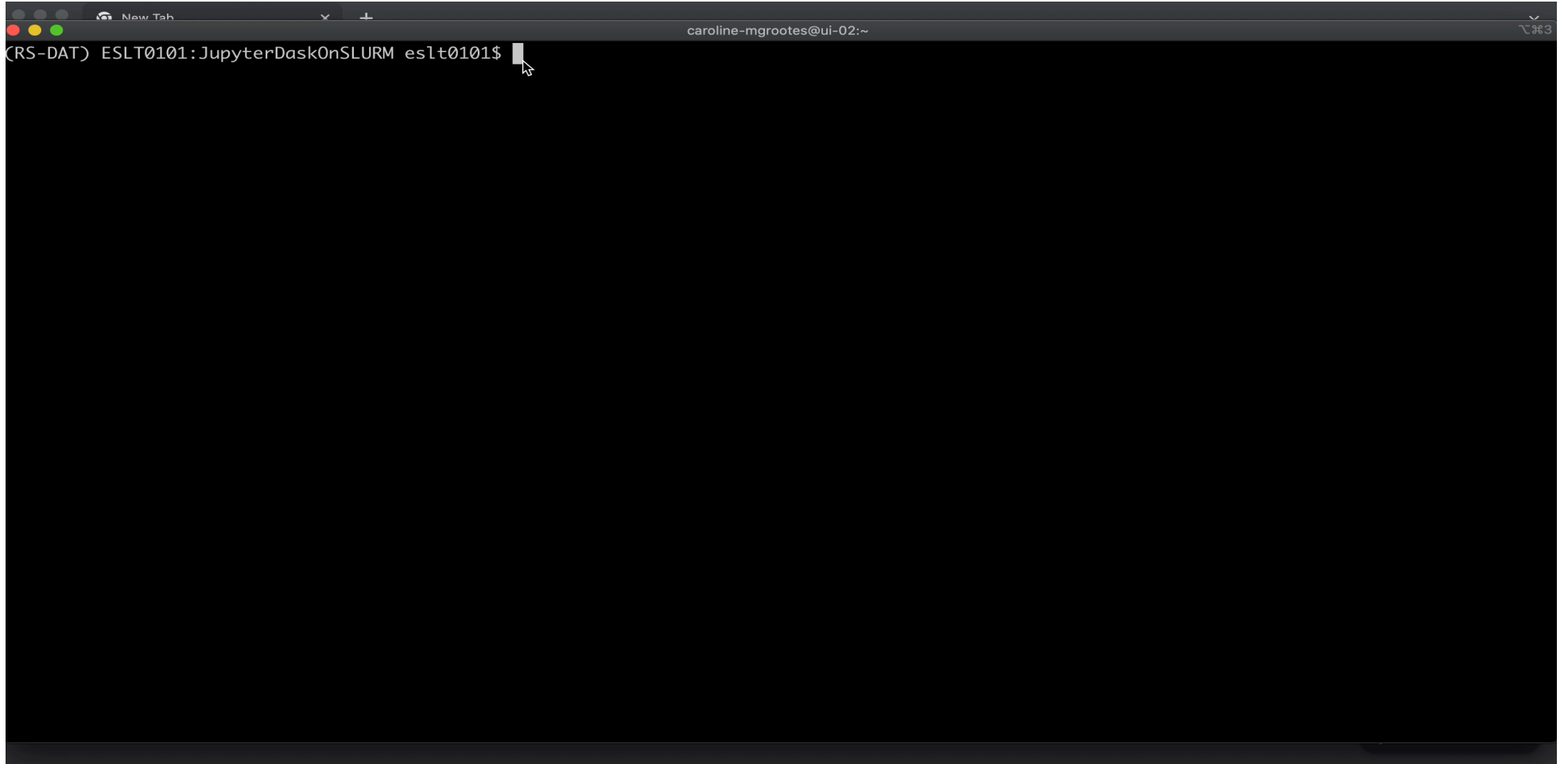


- JupyterLab instance with Dask and Git extensions, scalable Dask cluster, running on SLURM managed HTC/HPC system
- Basic idea
  - Launch JupyterLab server and Dask scheduler as long-running batch job
  - SSH port-forwarding to connect to JupyterLab server
  - Launch workers as short-lived batch jobs (fast thru queue)



Ensure ease of use by abstracting details from user (but configurable if desired)

- Set up your environment.yaml file and you're good to go locally
- Launch via command line
- That's it!



A terminal window screenshot showing a shell prompt. The window title is "New Tab" and the user is "caroline-mgrootes@ui-02:~". The prompt is "(RS-DAT) ESLT0101:JupyterDaskOnSLURM eslt0101\$".





The screenshot shows a JupyterLab notebook in a Safari browser window. The notebook is titled "02-compute-spring-index.ipynb" and is running on a Python 3 (ipykernel) environment. The interface includes a menu bar (File, Edit, View, Run, Kernel, Git, Tabs, Settings, Help), a toolbar with icons for file operations and execution, and a central code editor.

The code editor contains the following Python function:

```
def add_mean_plant_layer(outdate):
    """
    Average the spring index date over plant species and add the mean
    as a new layer.
    """
    mean = outdate.mean(dim="plant", skipna=False).round()
    mean = mean.expand_dims(plant=["mean"])
    return xr.concat([outdate, mean], dim="plant")
```

Below the function, there is a section titled "2.4 Open the input catalog" with the following text:

The input variables (minimum temperature, maximum temperature and day length duration) are extracted from the Daymet catalog, which we have downloaded earlier as a STAC catalog (see [this notebook](#)). In order to get access to the data we load the catalog:

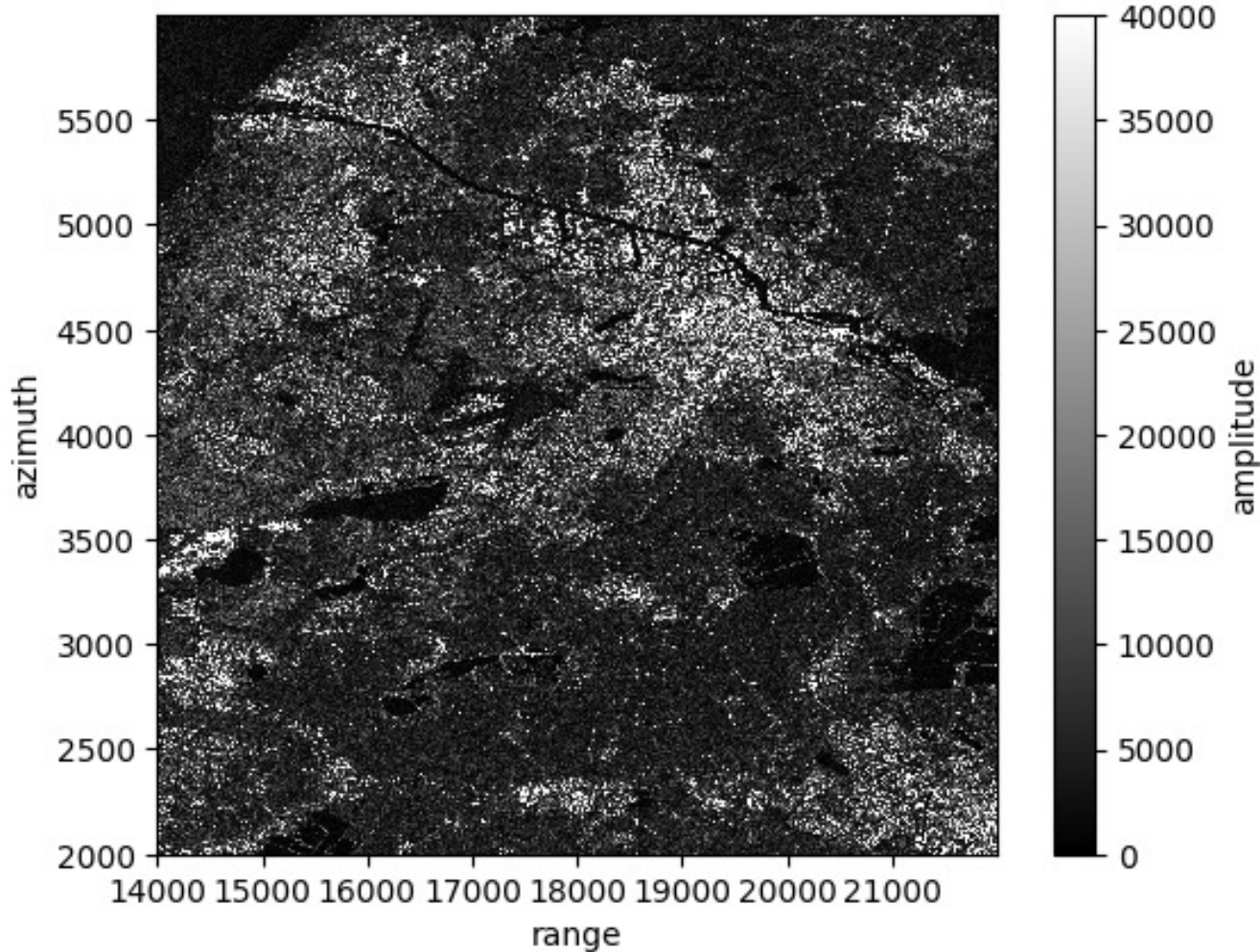
```
[ ]: catalog = pystac.Catalog.from_file(catalog_urllpath)
```

In addition to providing links to the data, the catalog provides all the dataset's metadata, which we use e.g. to convert the bounding box from latitude/longitude degrees to the dataset's coordinate reference system (CRS):

```
[ ]: # Extract information about input CRS from metadata
_item = next(catalog.get_all_items())
proj_json = _item.properties["proj:projjson"]
crs_lcc = pyproj.CRS.from_json_dict(proj_json)

# Set up CRS converter
transformer = pyproj.Transformer.from_crs(
    crs_from="EPSG:4326",
    crs_to=crs_lcc,
    always_xy=True,
)
```

The bottom status bar shows "Simple" mode, "0" lines of code, "1" cell selected, "main" environment, "Python 3 (ipykernel) | Idle", "Mode: Command", "Ln 1, Col 1", and the file name "02-compute-spring-index.ipynb".

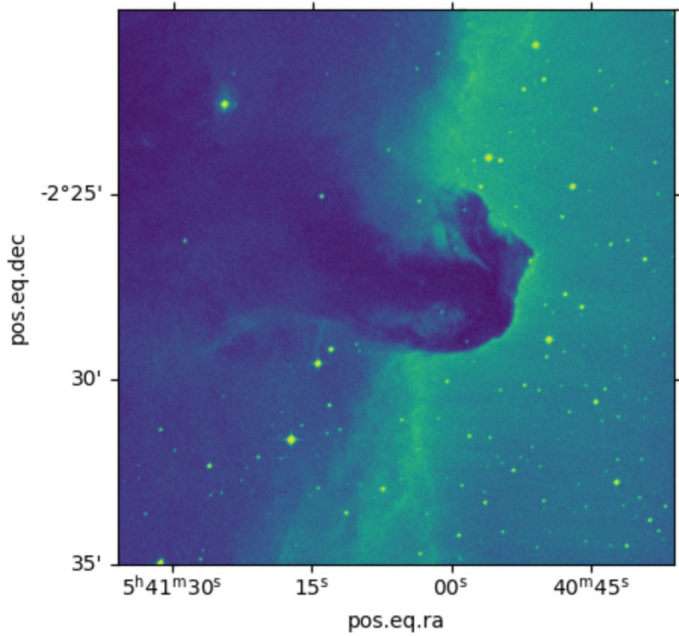


**Mean Reflectance Map of Amsterdam**

Images processed: 100  
Images size: ~500 GB

RS-DAT Runtime: ~10 mins  
Previous Runtime: ~1 week

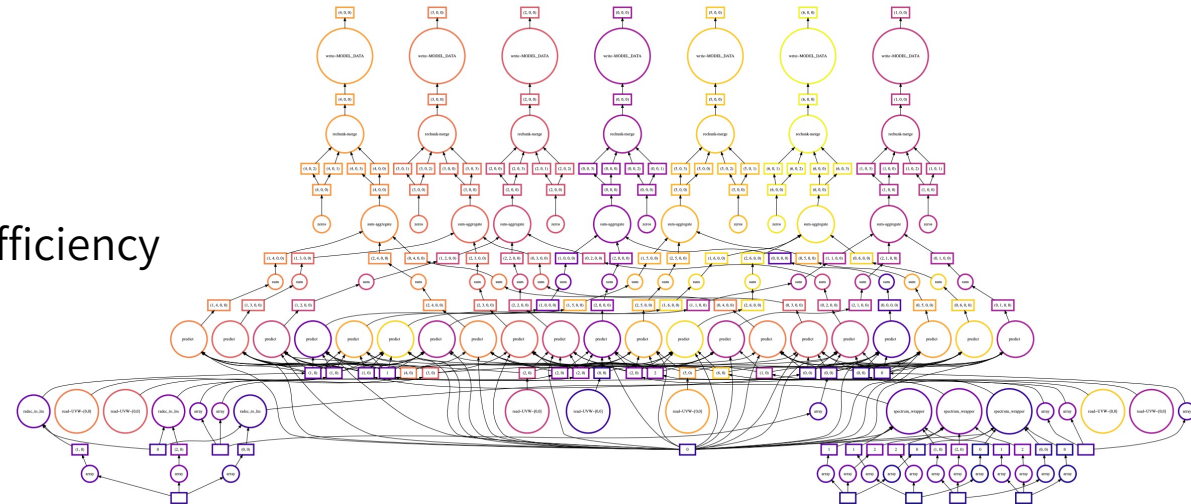
- Off-the-shelf use of DAT
- Integration with SARXarray
- Simple HTC parallelization of established workflow – orders of magnitude speed



Horse Head nebula from FITS w/ ndcube

- Dask backed image/IFU data processing w/ ndcube
- Large array of (nascent) radio astronomy tooling based on dask; dask-ms, SKA Fourier transform, etc.
- LSDB & HiPsCat; Dask backed astronomical source catalog (cross-)analysis at PB scale for LSST

- Packages are python ecosystem and dask based
- Require significant compute and storage resources for efficiency
- Should work off-the-bat with RS-DAT.
- We be interested and happy to help/think along



Prediction of model visibilities using dask-ms. Perkins et al 2021

- RS-DAT provides easy to deploy scalable analysis
- Full integration with PyData/Python ecosystem
- Full integration with mass storage dCache
- Fully automated deployment on SURF infrastructure. No further configuration needed. Customized (local) deployments also simple -> DelftBlue
- Work in progress – HPC and SURF research cloud integration, ML for RS support, ...
- What do you need? Could you use this?



**What would you like/need? Come talk to us!**  
[team-atlas@esciencecenter.nl](mailto:team-atlas@esciencecenter.nl)

**Let's stay in  
touch!**



[www.eScienceCenter.nl](http://www.eScienceCenter.nl)



[m.grootes@esciencecenter.com](mailto:m.grootes@esciencecenter.com)



+31 (0)20 460 4770

e