maap-docs

NASA MAAP Team

Nov 06, 2023

CONTENTS

1 Getting Started 1

2 Science Examples 33

3 Technical Tutorials 135

4 System Reference Guide 215

5 Release Notes 257

6 Indices and tables 259

CHAPTER

ONE

GETTING STARTED

1.1 About the Multi-Mission Algorithm and Analysis Platform (MAAP)

The MAAP platform is designed to combine data, algorithms, and computational abilities for the processing and sharing of data related to NASA's GEDI, ESA's BIOMASS, and NASA/ISRO's NISAR missions. These missions generate vastly greater amounts of data than previous Earth observation missions. There are unique challenges to processing, storing, and sharing the relevant data due to the high data volume as well as with the data being collected from varied satellites, aircraft, and ground stations with different resolutions, coverages, and processing levels.

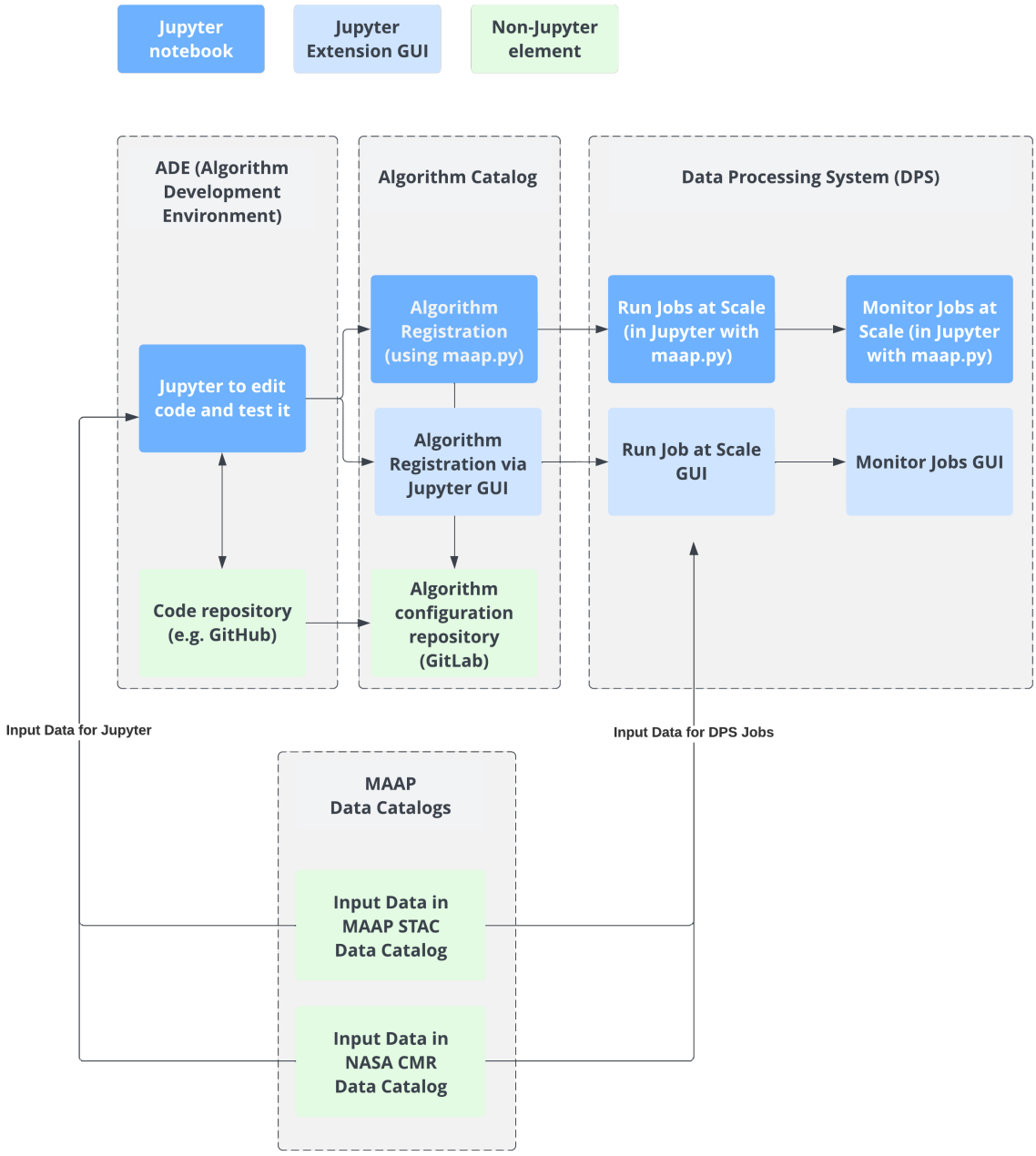
MAAP aims to address unique challenges by making it easier to discover and use biomass relevant data, integrating the data for comparison, analysis, evaluation, and generation. An algorithm development environment (ADE) is used to create repeatable and sharable science tools for the research community. The software is open source and adheres to ESA's and NASA's commitment to open data.

NASA and ESA are collaborating to further the interoperability of biomass relevant data and metadata. Tools have been developed to support a new approach to data stewardship and there is a data publication workflow for organizing and storing data and generating metadata to be discoverable in a cloud-based centralized location. The platform and data stewardship approaches are designed to ease barriers and promote collaboration between researchers, providers, curators, and experts across NASA and ESA.

This guide aims to help users get started with using the platform for searching, visualizing, accessing, processing, querying, and sharing biomass relevant data to the MAAP. These data, collected from satellites, aircraft, and ground stations, are organized into collections and granules. Collections are a grouping of files that share the same product specification. Granules are the individual files which are independently described, inventoried, and retrieved within a collection. Granules inherit additional attributes from their containing collection. Explanations of the various functions available in MAAP to use in the ADE will also be explored.

1.2 An overview of the MAAP platform

The MAAP is a cloud-based system to write science-analysis code and then run it at scale. This lets you keep all of the input and output data "in the cloud". It is composed of a few parts:



- The **Algorithm Development Environment (ADE)** is a tool that helps with the development of algorithms in a consistent, standardized environment that helps with the development and testing of algorithms and facilitates large scale data processing. MAAP's primary user interface is Jupyterlab, where code is written and tested before pushed to the large scale data processing system. Code is stored and checked out from Git-based repositories,

including Github and MAAP's own code repository subsystem.

- The **Data Processing System (DPS)** is where registered algorithms (see Algorithm Catalog) can be run at scale in the cloud. The MAAP system provides a Jupyter GUI to run Jobs, or the maap.py library can be used to run a batch of Jobs in a loop using Python. The DPS also has monitoring capabilities, and again the MAAP system provides a Jupyter GUI to help monitor Jobs. This can also be done using maap.py in Python.
- The **Algorithm Catalog**, where your algorithms from the ADE can be registered and compiled for use by the DPS. The MAAP system provides API and GUI tools to help you register and view your algorithms.
- The **Code Repository** is a git-based repository to store user code. It is also used to store the configuration files necessary for building algorithms to store in the algorithm catalog and for execution in the DPS.
- Input data comes from a few **Data Catalogs**. Currently there is a MAAP STAC Catalog and the NASA CMR Catalog. More information can be found in the search tutorials section.

1.3 Setting up your account and workspace

Learn how to sign up for an account and access the MAAP.

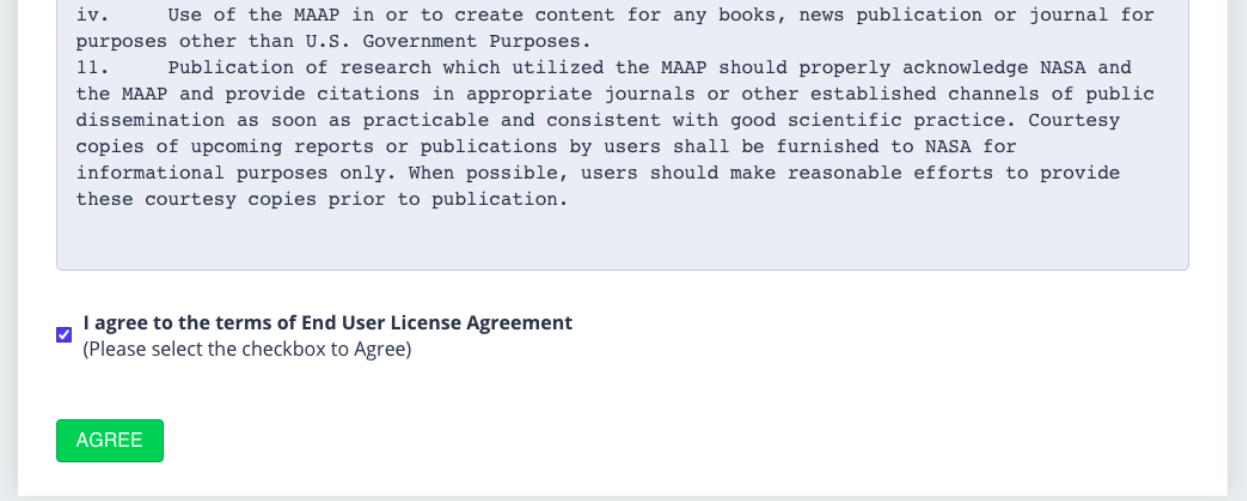
1.3.1 Signing up for an Earthdata Login account

The MAAP offers accounts for NASA users through Earthdata Login. Before accessing the MAAP as a NASA user, you will need to create an Earthdata Login account. Anyone can register for an Earthdata Login profile here: https://urs.earthdata.nasa.gov/users/new.

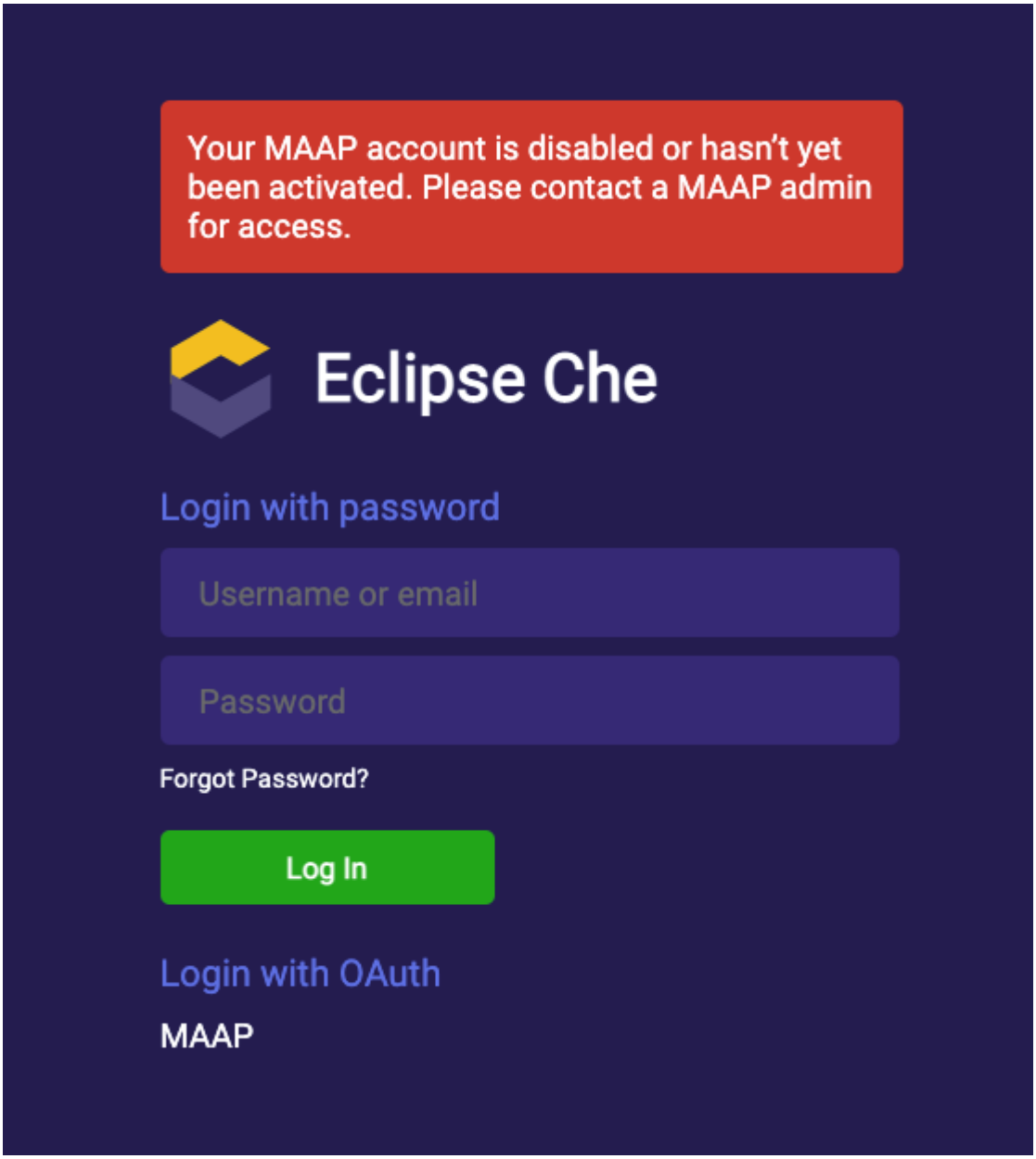
1.3.2 Signing up for a new MAAP account

Once registered, you can register for a MAAP account by navigating to the MAAP ADE at http://ade.maap-project.org. On your first visit, select the "URS" login button shown here:

If this is your first visit to the MAAP, you will be asked to agree to the MAAP Terms of Use:



Once registered, you should be redirected back to the MAAP ADE showing a disabled account message similar to this:

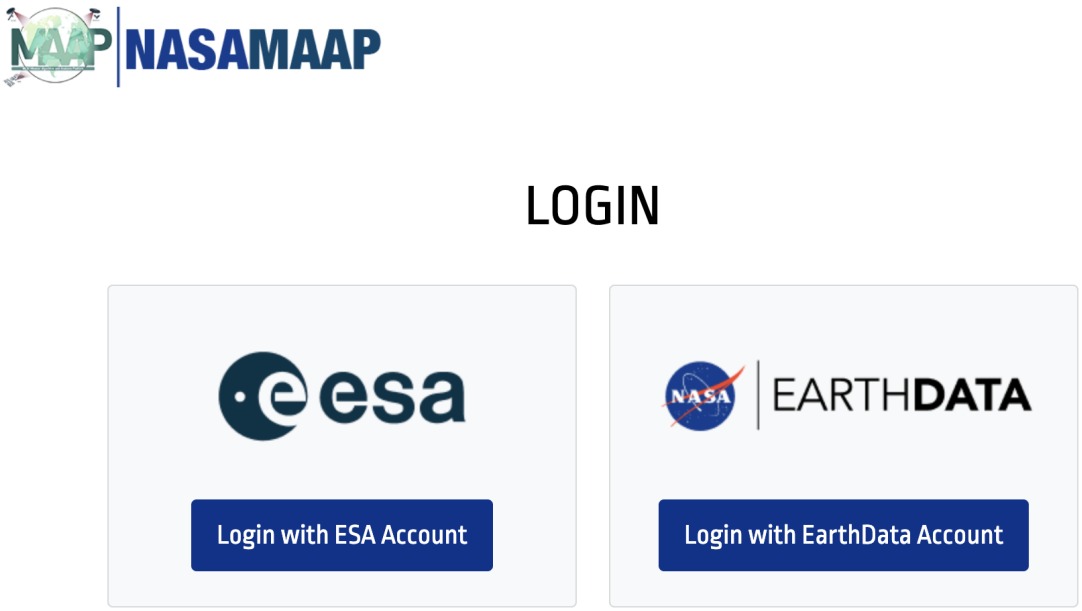


At this point, a MAAP administrator will approve your account, which will grant you access to the MAAP ADE. Access is only granted to known users in the biomass science community and other projects directly related to MAAP. To check on the status of your pending account, contact the MAAP team at support@maap-project.org.

Note: Once your MAAP account is approved, you will receive an email notification using the address of your Earthdata Login account to let you know that your access is enabled.

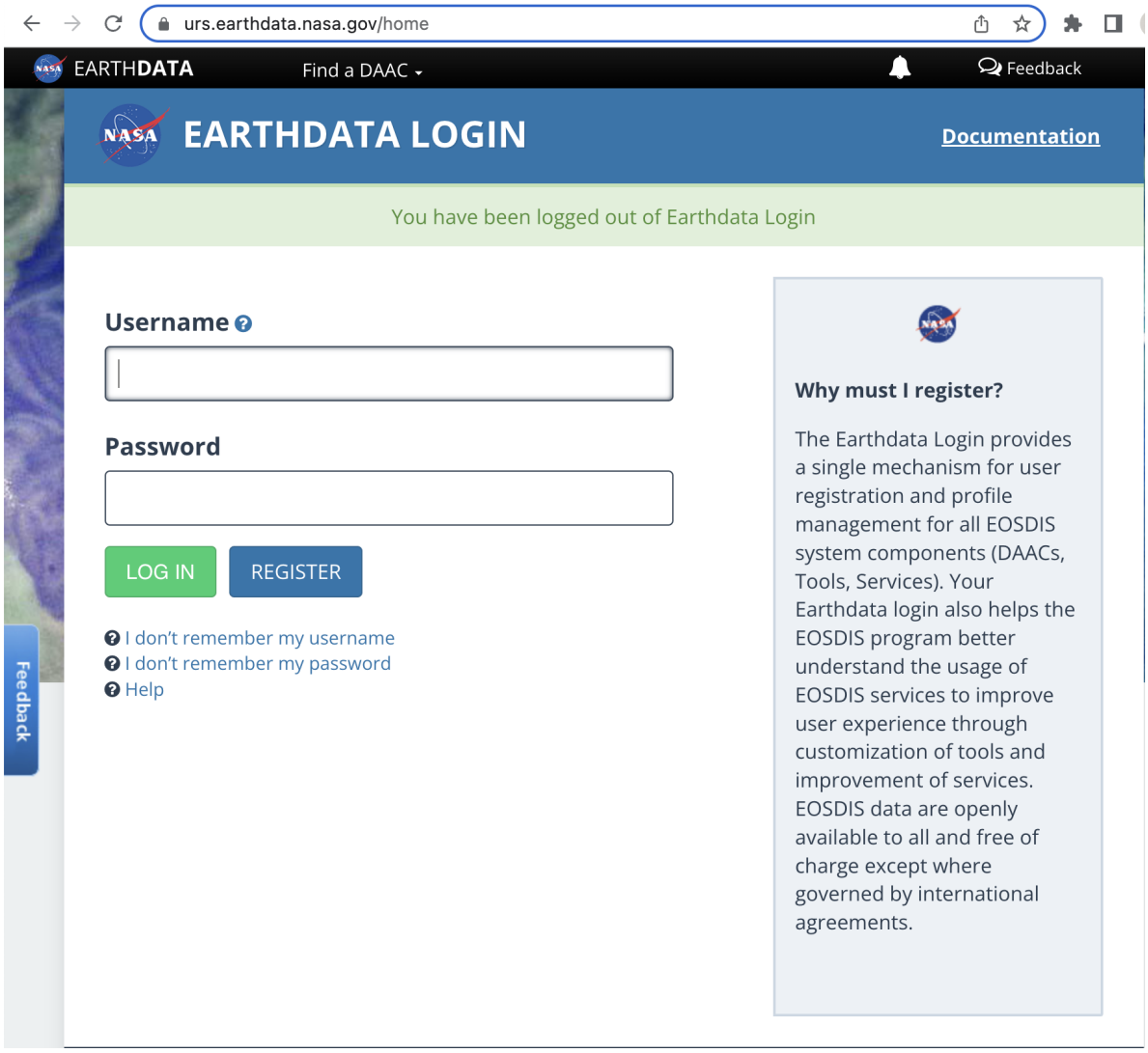
1.3.3 Logging in

1. Navigate to https://ade.maap-project.org/ in Chrome or Firefox. You should be redirected to a page that looks like

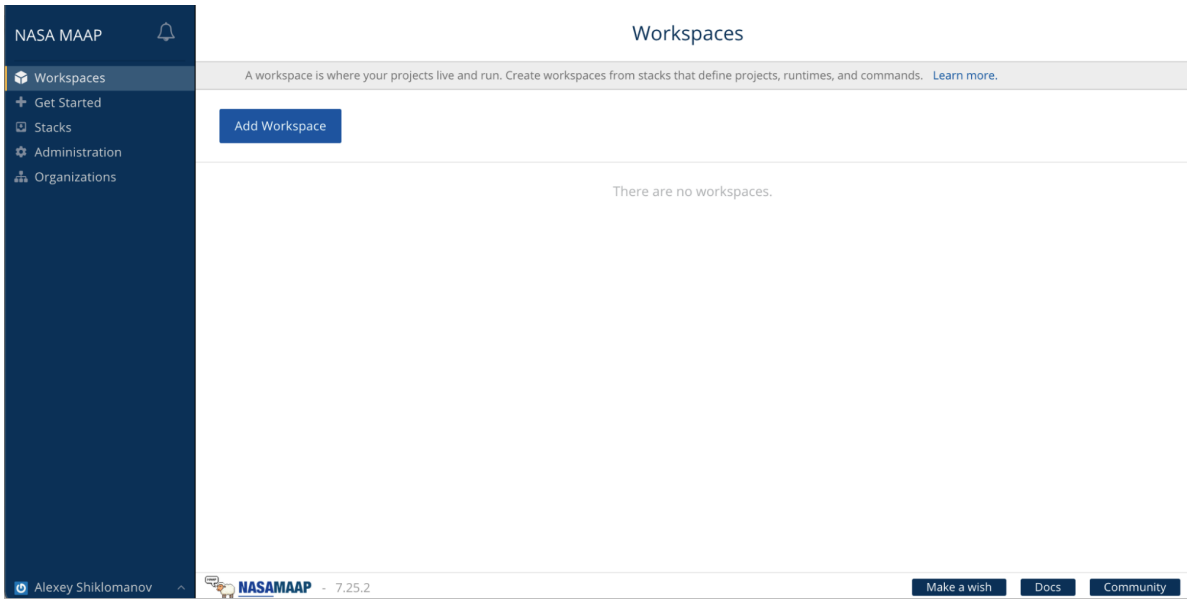


   this:

2. Click the "Login with EarthData Account" button. If this is your first time logging in, you should be redirected to an EarthData Login page that looks like this:



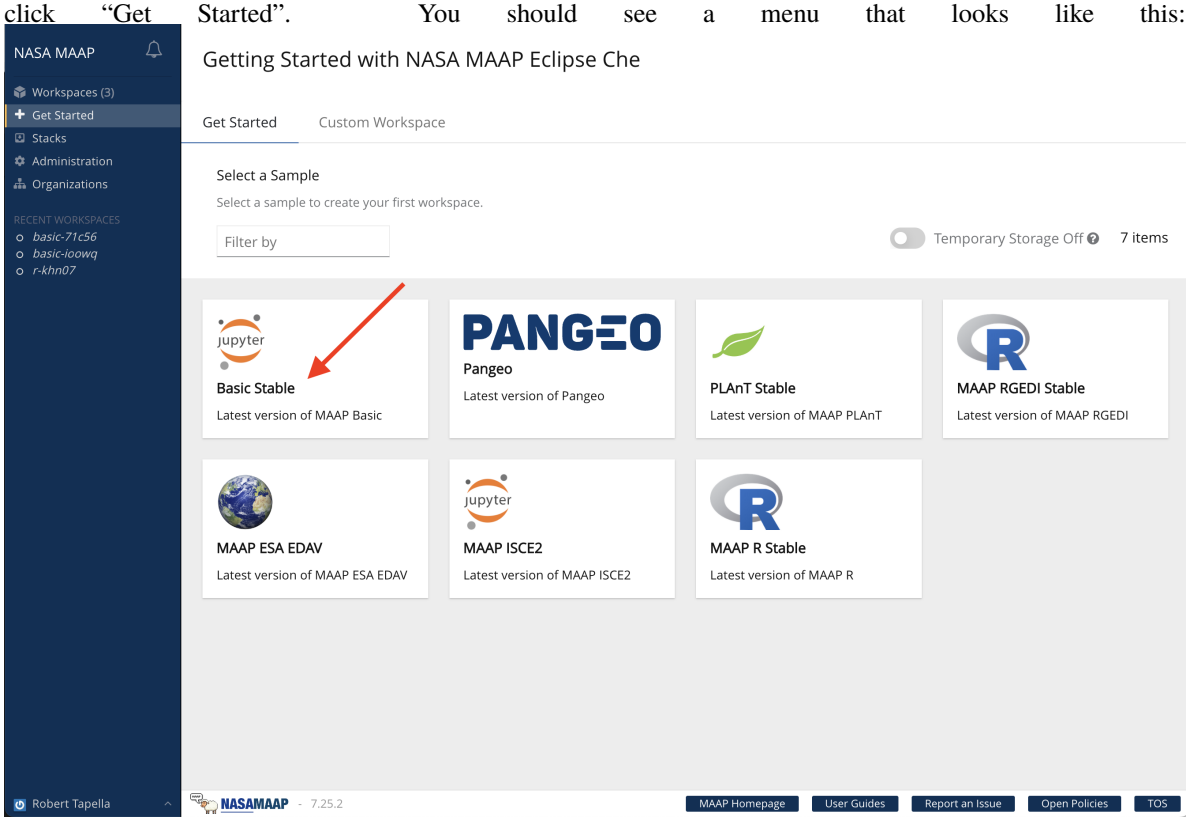
3. Enter your "EarthData Login" account credentials here and click "Log in". You should see a temporary page that says "Redirecting", followed by the MAAP showing your Workspaces (which will be empty to start):



1.3.4 Creating a workspace

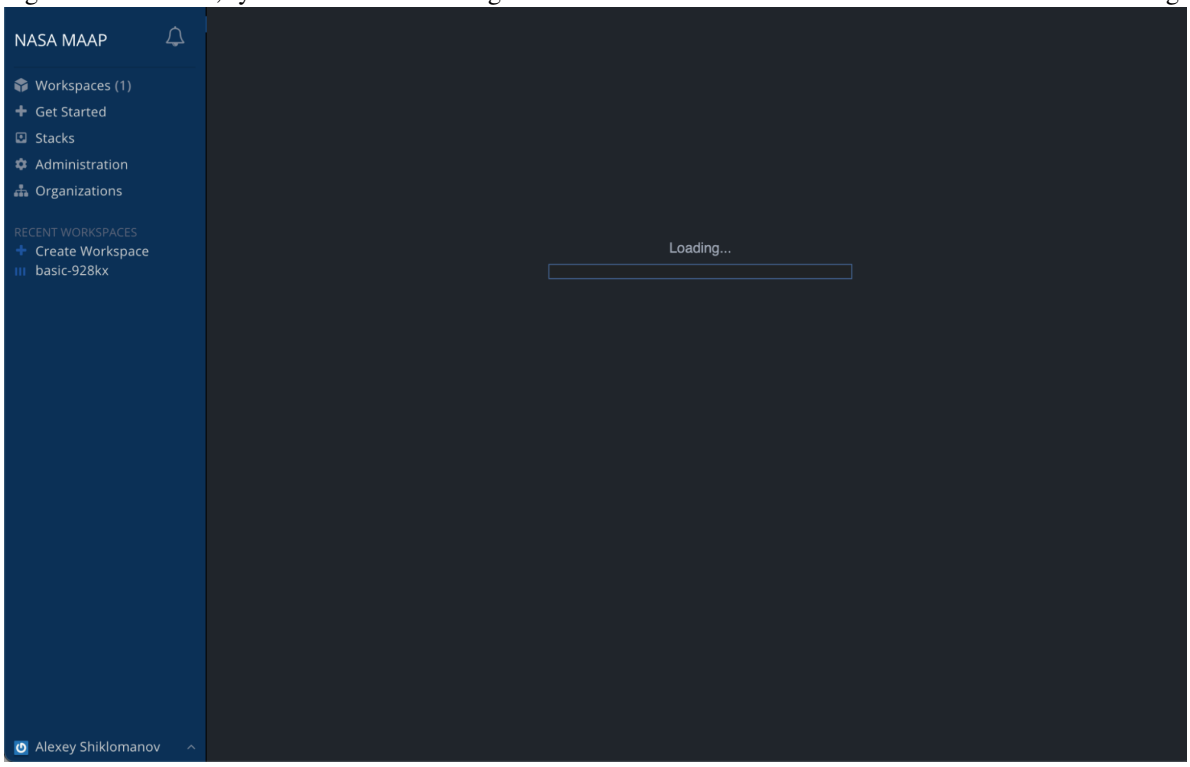
Workspaces are effectively a JupyterLab "computer in the cloud". To get started with Jupyter you need to create a workspace.

1. In the top-left corner of the MAAP dashboard, under "NASA MAAP", click "Get Started". You should see a menu that looks like this:

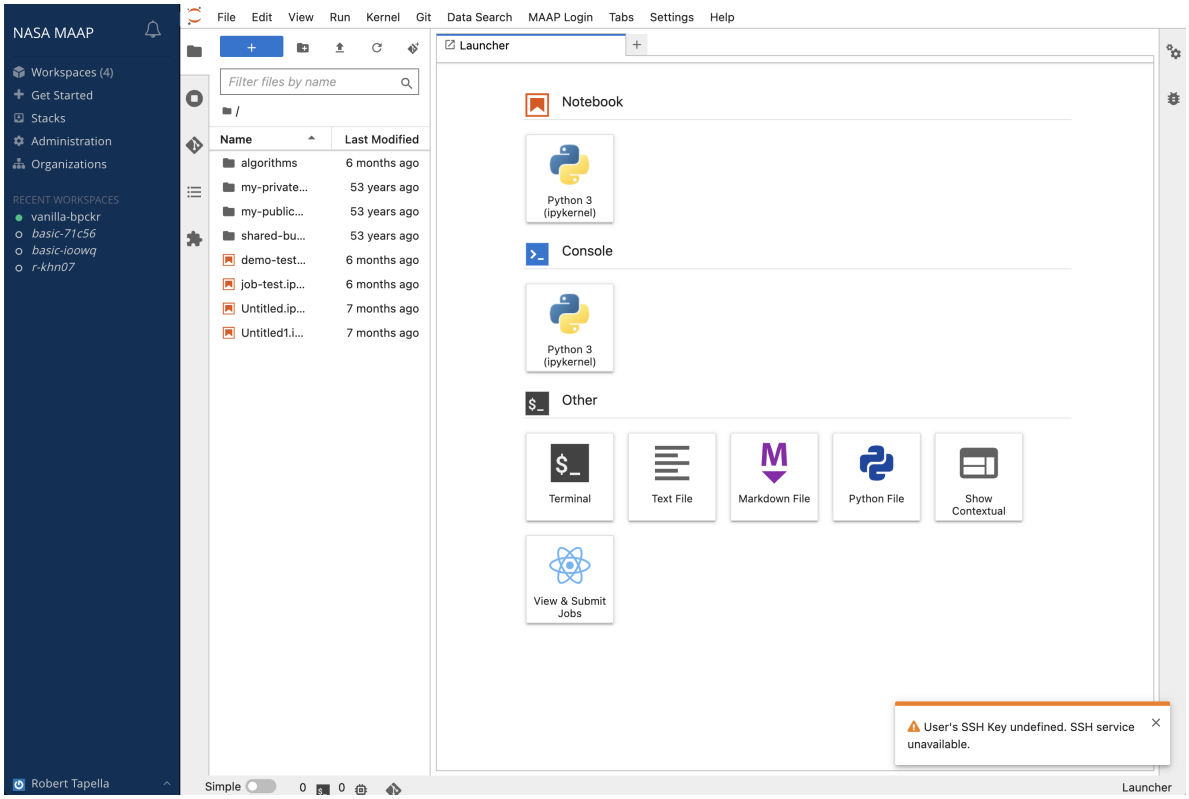


2. Select "Basic Stable". This is called a "Stack" and represents a type of cloud compute environ-

ment that will be set up. If you are interested in seeing more about each Stack, the adjacent link in the left-hand area is where you can see the configuration of each Stack in detail. After choosing "Basic Stable", you will see a loading screen that looks like this – wait for it to finish loading.

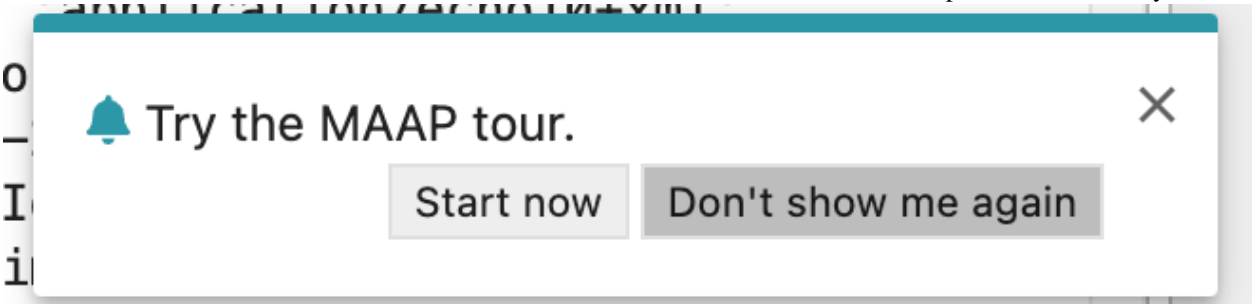


3. Once the workspace has loaded, you should see a Jupyter interface that looks like this (note: You will see fewer environments and items in your root directory — this is normal! You may also see some notifications in the bottom right that look like errors about SSH Keys and other things; that is normal as well. You will also see one asking if you would like to take a guided tour.).



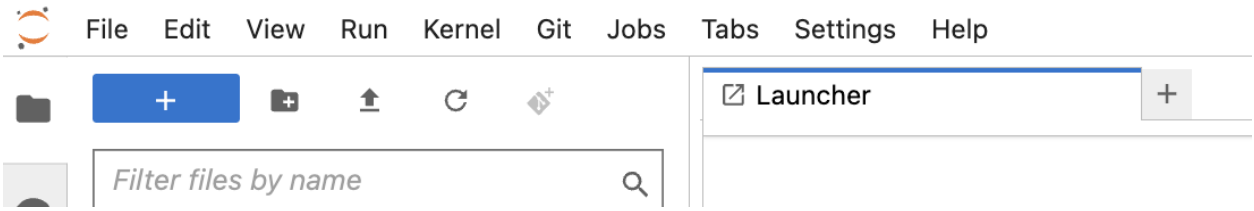
1.3.5 Jupyter Interface overview

When you first log in you may see a notification in the bottom right about a guided tour. Feel free to view the tour, which will give you a quick overview of the Jupyter user interface. You can also find the MAAP Tour in the Help menu at any time.



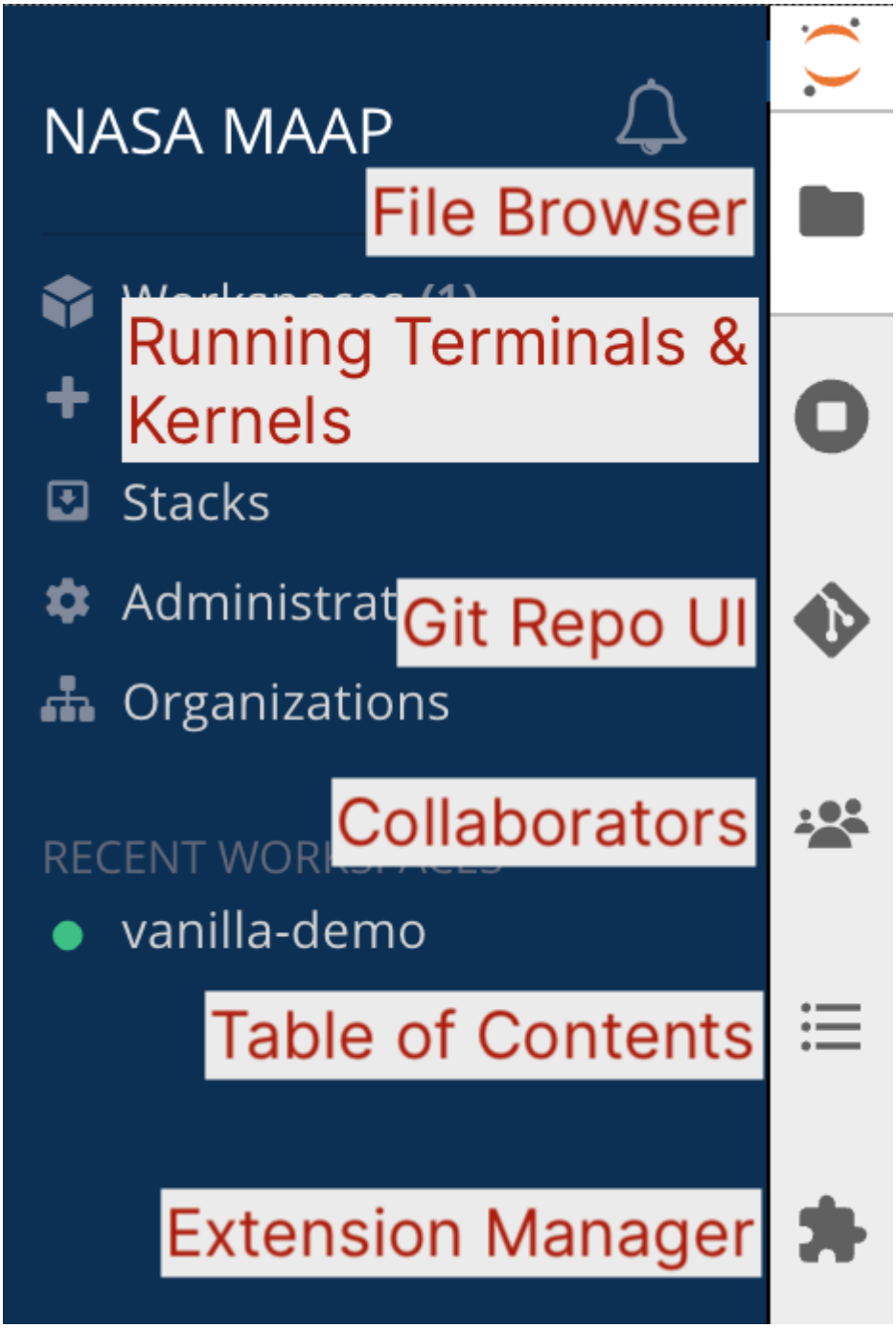
In addition to typical JupyterLab menu bar and sidebar configuration:

MAAP Jupyter Menus



- Git: Open repo in terminal, init, or clone repo.
- Jobs: Users may submit jobs through the submit tab and view their jobs through the view tab.
- Help: The help menu has several customized extensions and references to the MAAP documentation.

MAAP Jupyter Sidebar



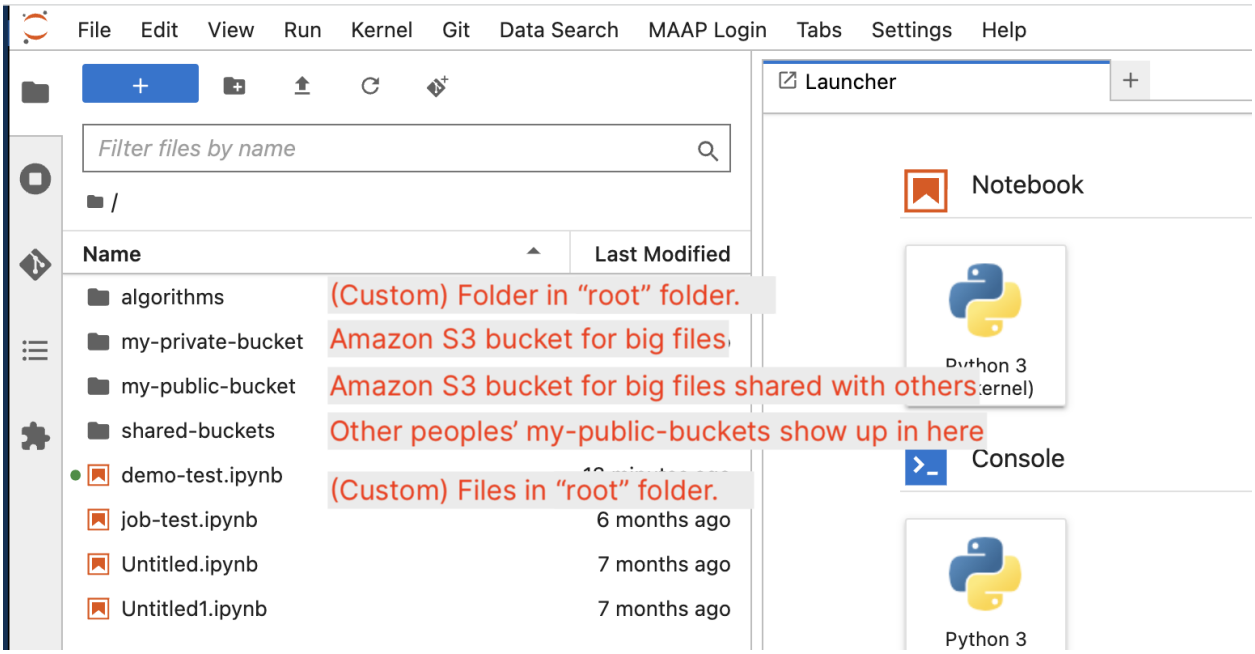
- File Browser
- Running Terminals & Kernels
- Git Repo Interface (if this folder is a Git repo)
- Collaborators
- Table of Contents

- Extension Manager

MAAP Blue Sidebar

- Workspaces: See workspaces, share them, as well as configure settings
- Stacks: See available platforms for workspaces & required memory
- Administration: Control the configuration & policies for your installation.
- Organizations: allow groups of developers to collaborate with private & shared workspaces. Resources & permissions are controlled & allocated within the organization by admin.
- Profile (bottom, labeled with your name): See account info, logout

1.3.6 MAAP Storage Options



My root folder (fast cloud storage)

- Your Jupyter home directory (~) is mounted to `/projects`. Files in here persist across sessions and exist across your workspaces.
  - Use this for code-related items, smaller data storage
  - Git is more likely to behave predictably here compared to other storage
  - This is also the place to make persistent conda environments (covered in another section), but make sure to not make a conda env inside a git-tracked folder, or if you do add it to the .gitignore. If git is tracking an env, it could cause your workspace to crash.
- Uses local (to Jupyter) file system; generally faster and more reliable for "normal" file operations, but expensive

Large file storage: my-private-bucket

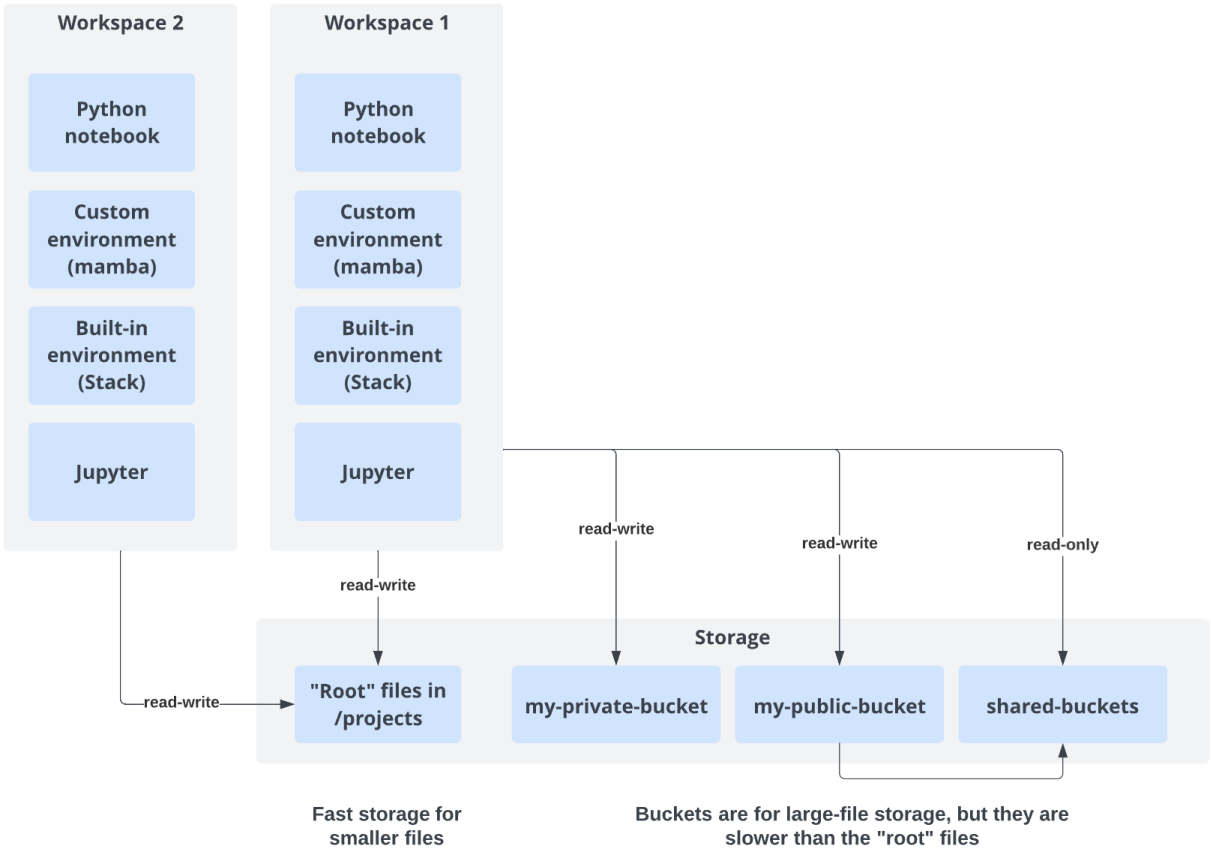
`~/my-private-bucket` is an S3 bucket with persistent storage, but accessible only to you and others in a shared workspace.

- Use for large data storage
- It will be slower than the root folder to copy and move files, which is why it is not ideal for storing smaller files that need to be read or written quickly

Sharing files: my-public-bucket and shared-buckets

`~/my-public-bucket` is an S3 bucket with persistent storage. It is the same as `~/shared-buckets/<my_username>/` — anything you put in here will be accessible to other users via `~/shared-buckets/<my_username>` as a read-only file. Likewise, to find shared files from another user, look in `~/shared-buckets/<their_username>`.

- Use for large data storage for files that you want to share across workspaces



Mounting your MAAP workspace on your local computer

If you prefer to work on your local computer, or to drag-and-drop copy files from your computer to/from MAAP, you access the workspace via SSH. The *process for doing this* is in the system guide.

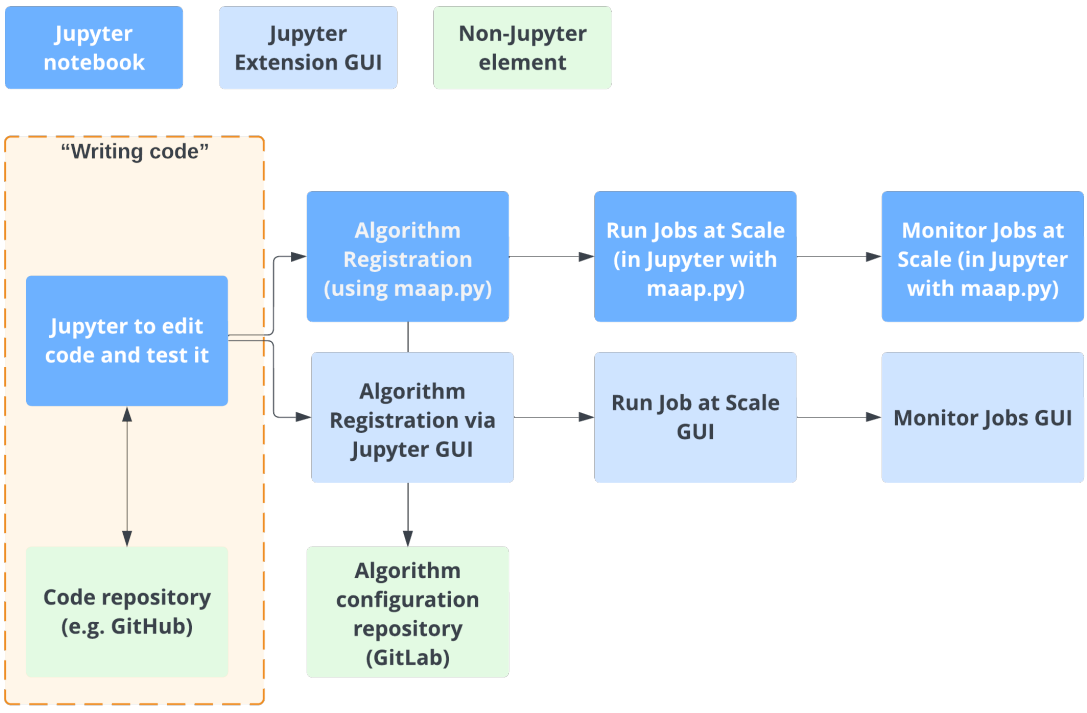
1.4 Writing and Managing Code with MAAP

Writing and editing code in the MAAP is done in a Jupyter workspace.

Code is version-controlled using git, which may be GitHub or MAAP GitLab. Jupyter also provides a GUI widget to help with code push/pull as a sidebar tool. Git is intended to help with collaborative code development and version-control.

Note: If you have not used Github or git before, it is highly recommended that you get acquainted with it. For a quick reference to git commands there is a Git Cheat Sheet in a variety of languages.

To assist connections to the MAAP system from a Jupyter notebook, a helper library called `maap.py` provides Python-native calls to the underlying RESTful MAAP API. Often a separate Jupyter notebook is used to run and monitor jobs with API calls. **When working with Jupyter notebooks, a manual save must be done to create a checkpoint.** Checkpoints are an emergency backup of the notebook and are different than the automatic saving of the notebook.



1.4.1 Helpful Templates while developing Algorithms in MAAP

- This algorithm repository example is a good starting point for a new algorithm, as it contains the various accessory files that facilitate running the algorithm at scale
- Which templates will help you? Let the development or documentation team know!
- For example: conda.yml with some default packages, run_script.sh

1.4.2 Working with code repositories like GitHub and GitLab

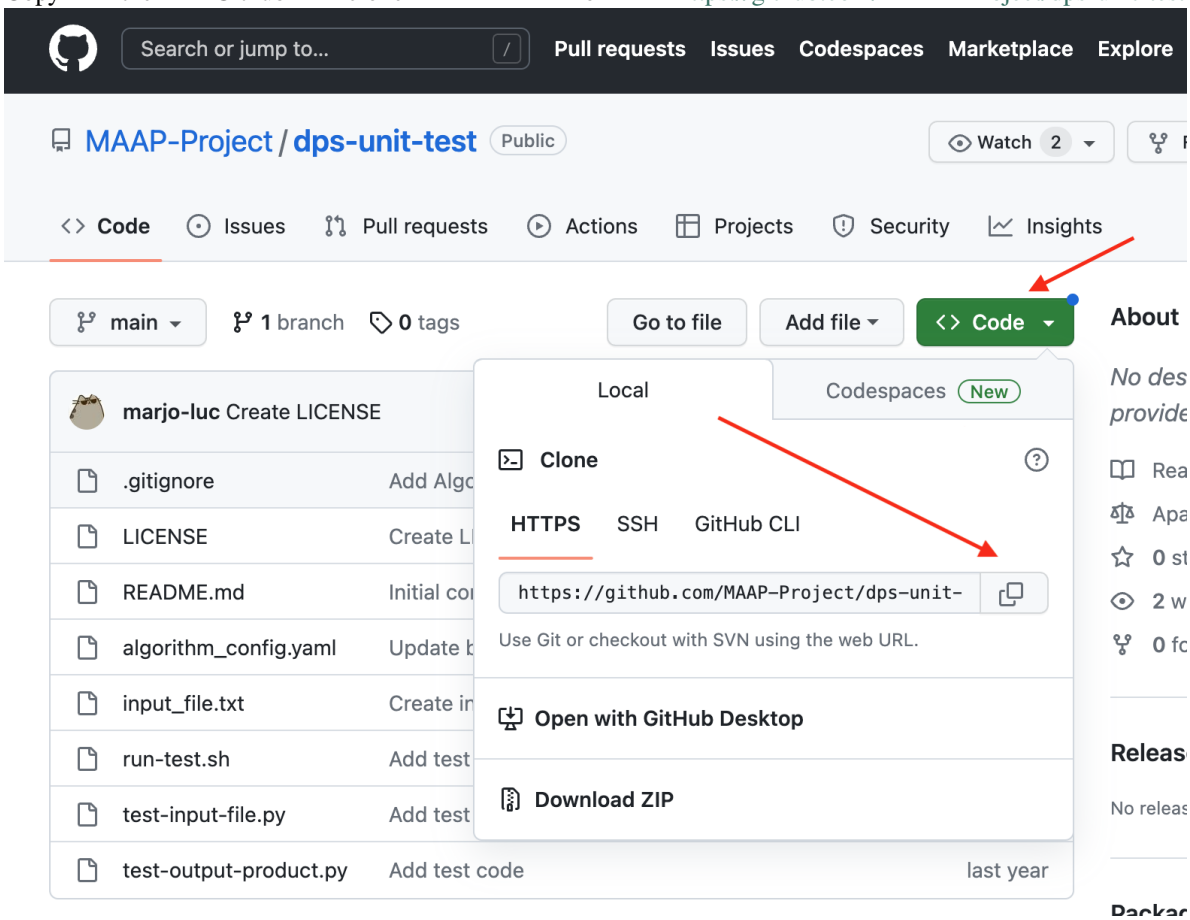
Connecting to Github using a Personal Access Token

Set personal access token: https://docs.github.com/en/authentication/keeping-your-account-and-data-secure/creating-a-personal-access-token

Clone a Repository with GitHub

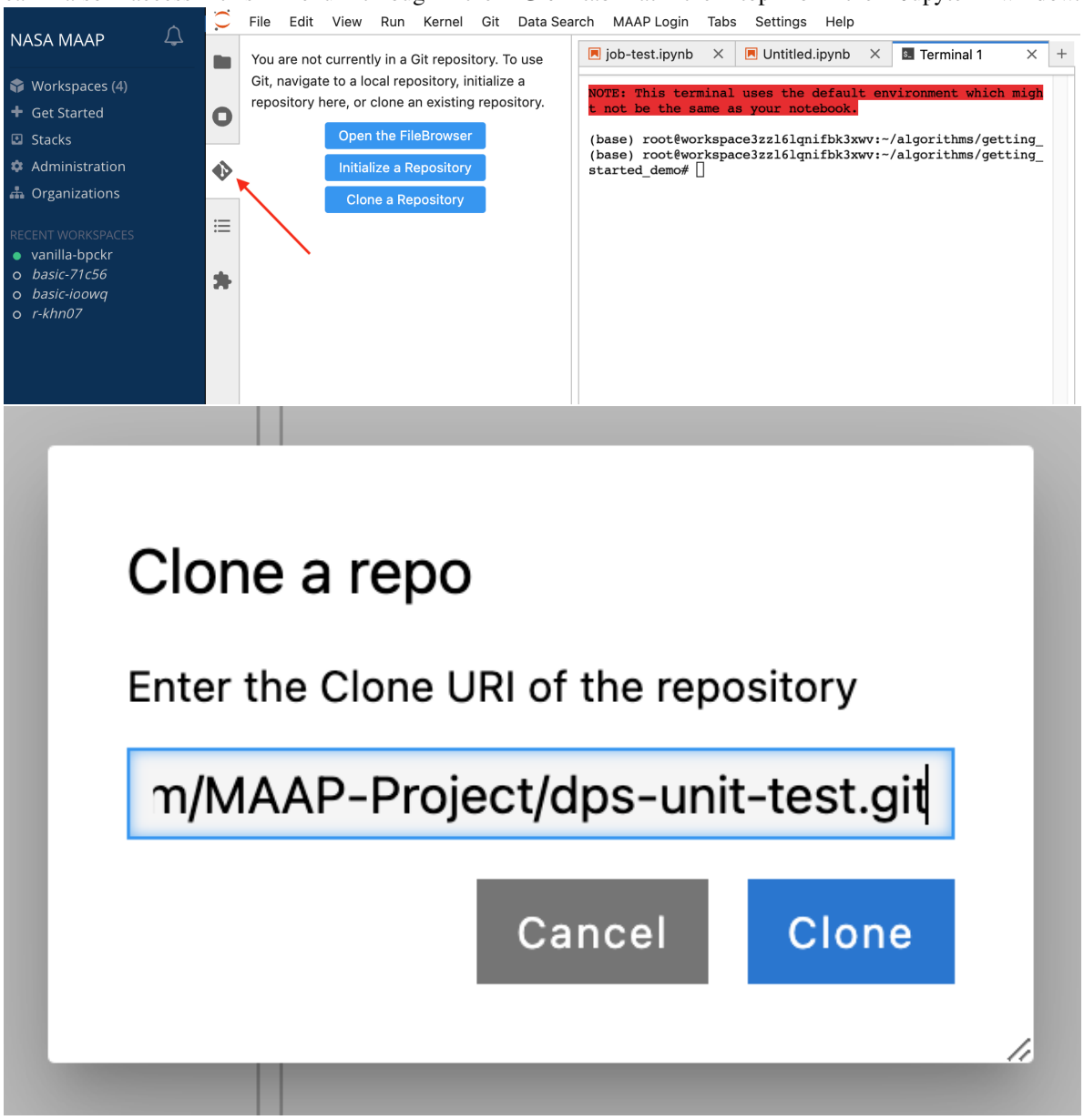
Here is an example repository you can use for this getting started guide: https://github.com/MAAP-Project/dps-unit-test

1. Copy the Github clone link from https://github.com/MAAP-Project/dps-unit-test

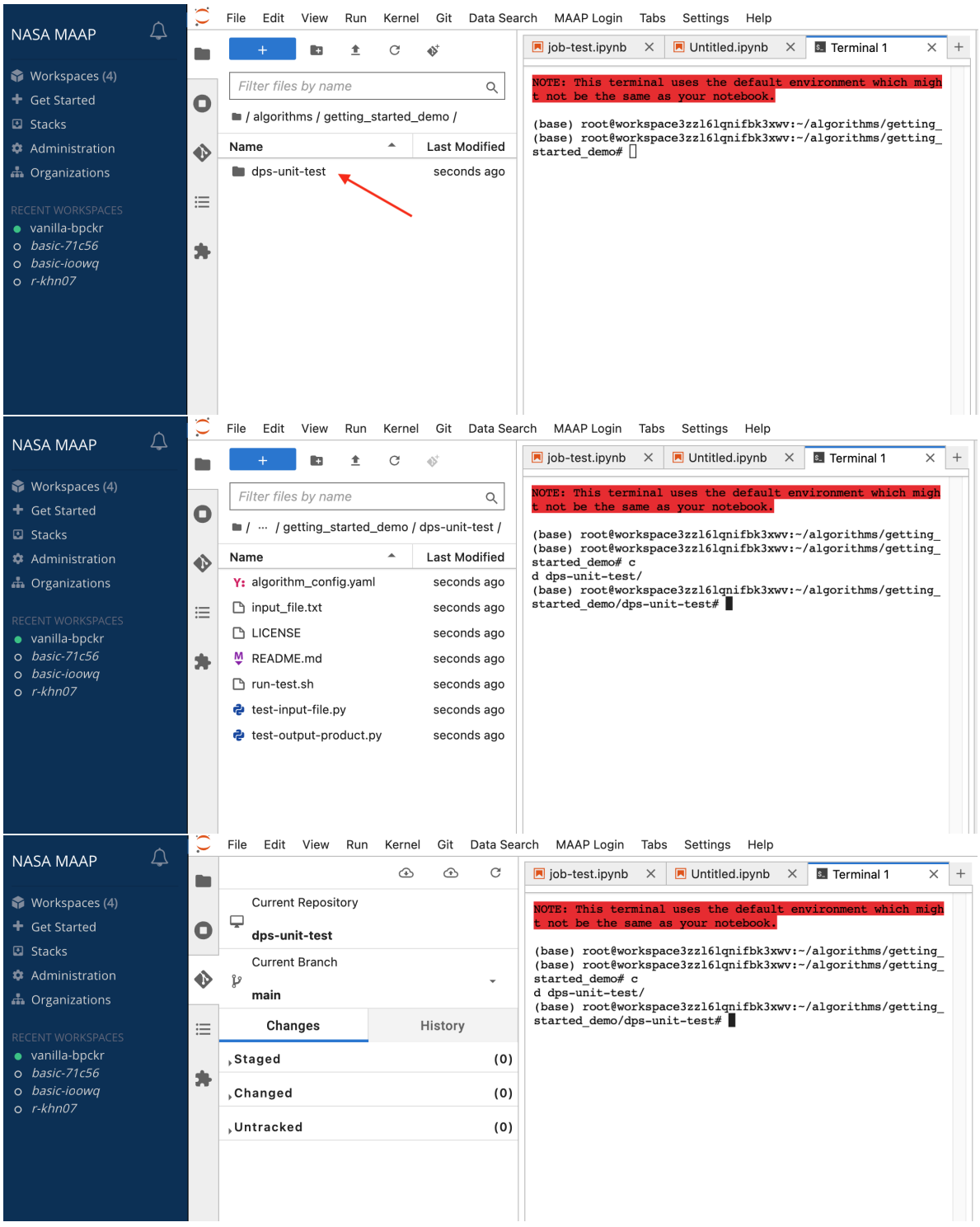


2. Open the built-in Jupyter Github UI to the left of the file browser. Choose "Clone a Repository" and paste in the .git link you copied from the Github repository. You

can also access this menu through the **Git** tab at the top of the Jupyter window.



3. You should see a new folder created with the repo you cloned. If you browse to that folder and open up the Jupyter Github UI again, it will show you some info about that repo.



4. If you want to make changes to the code and have your own copy of it to register, clone the code into a public repository in Github or in MAAP Gitlab.

To open the IPython Notebook, go to a section directory and double-click on appropriate ".ipynb" file. For more information about the using Git in Jupyterlab, see https://github.com/jupyterlab/jupyterlab-git .

The MAAP GitLab Code repository

After creating your MAAP account, you can create a code repository by navigating to the MAAP GitLab account at https://repo.maap-project.org. This GitLab account is connected to your ADE workspaces automatically when signing into the ADE.

You can then follow the same steps above to clone a repository from the MAAP GitLab.

1.4.3 Customizing your workspace environment

Your Jupyter workspace has a set of pre-installed libraries, depending on *which Stack you selected*. If you need libraries that are not pre-installed, we suggest using an environment manager; `conda` is pre-installed to help with this.

Full *documentation on configuring conda* may be found in the *System Reference Guide*.

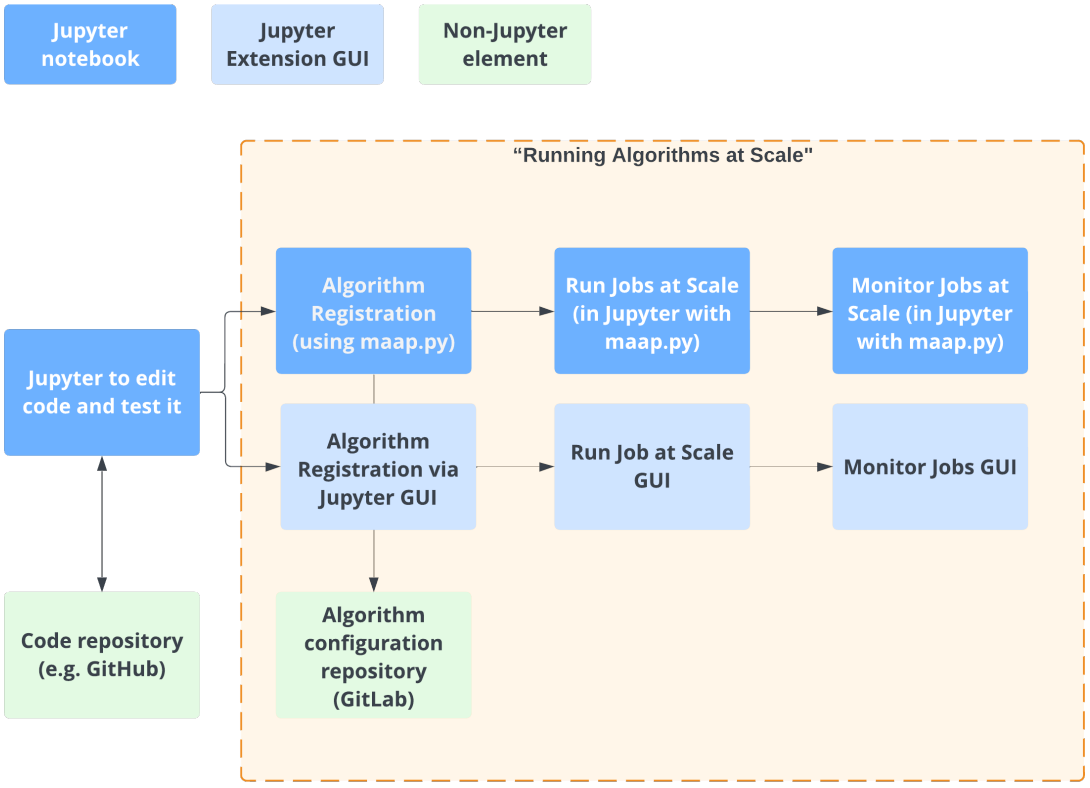
1.4.4 Using maap.py to access MAAP functionality from Python notebooks

The MAAP platform offers a variety of functionality to run and monitor large-scale processing jobs. Access to the functionality is gained via the underlying RESTful MAAP API. In a Python notebook, you will typically use this API via a helper library called `maap.py`, which will make using MAAP platform features easy, using Python syntax. For example, registering algorithms, running batches of jobs, monitoring jobs, or accessing data.

Much of the `maap.py` functionality is documented in the *Technical Tutorials section* and in-context in the *Science Tutorials*. The maap-py Github page has additional usage documentation.

1.5 Running Algorithms at Scale

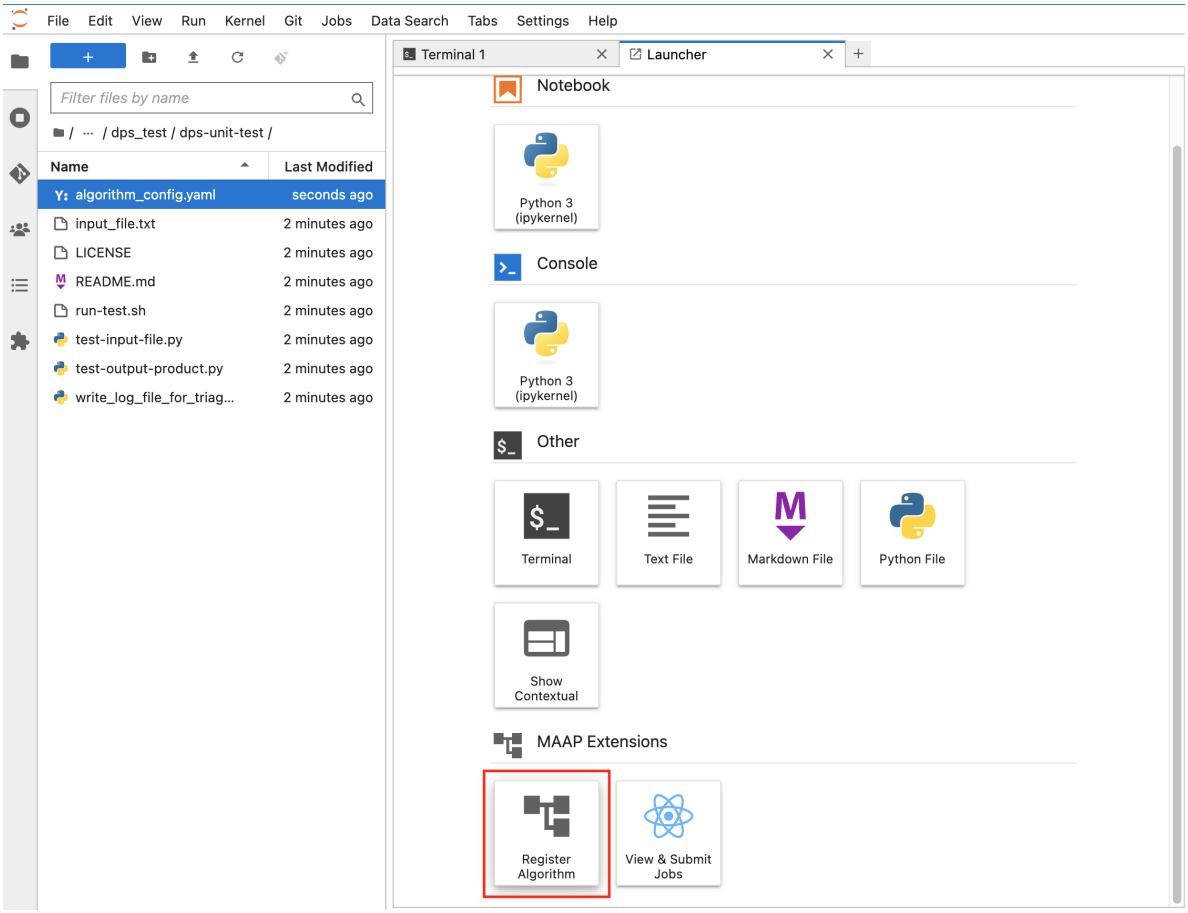
In order to run algorithms in the scaled-up cloud compute environment, they must first be "registered" in the Algorithm Catalog. This will make them available to other MAAP users, clearly define their inputs and outputs, and prepare them to be run easily in the Data Processing System (DPS).



1.5.1 Register an Algorithm

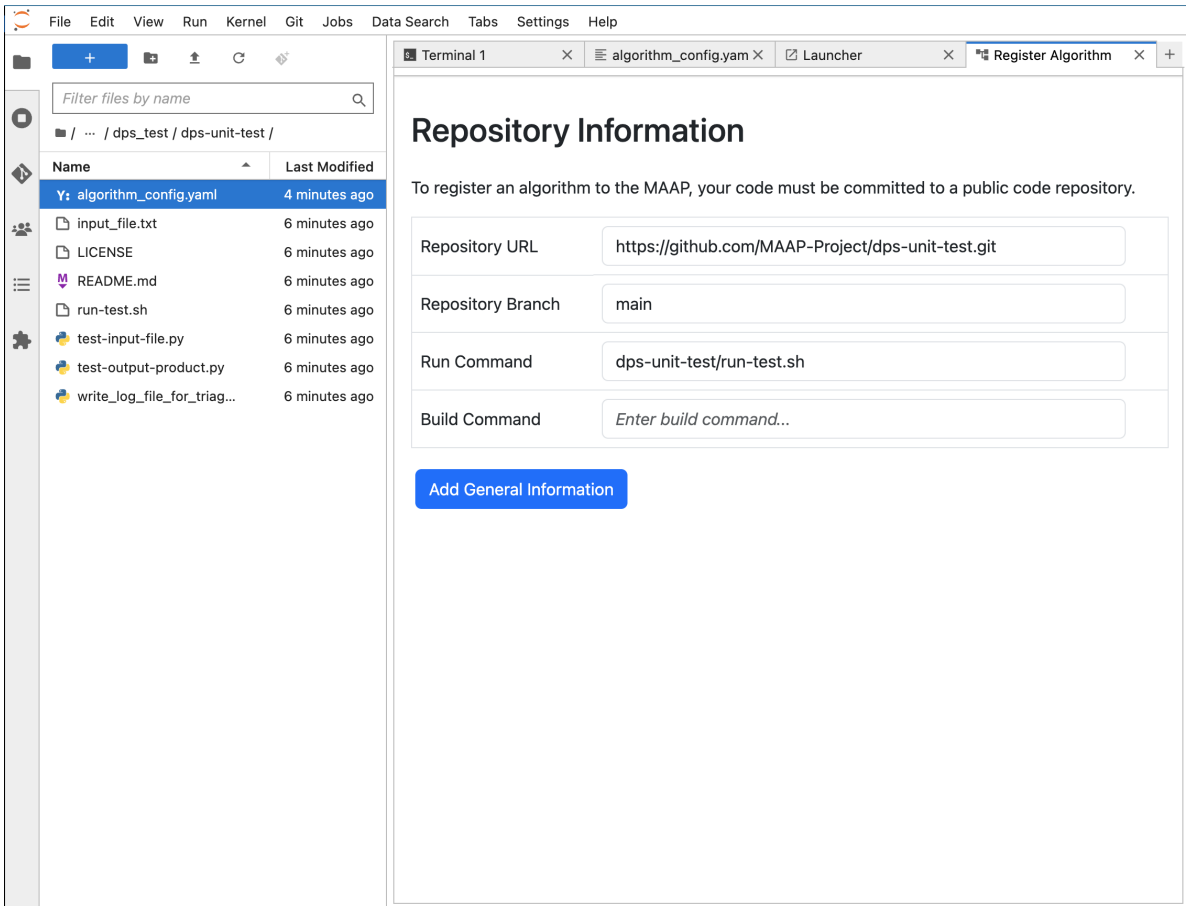
To register an Algorithm that can be run in the DPS, the code should be placed in a public Git repo (either Github or Gitlab).

1. Open the Launcher -> Register Algorithm tool in the MAAP Extensions section



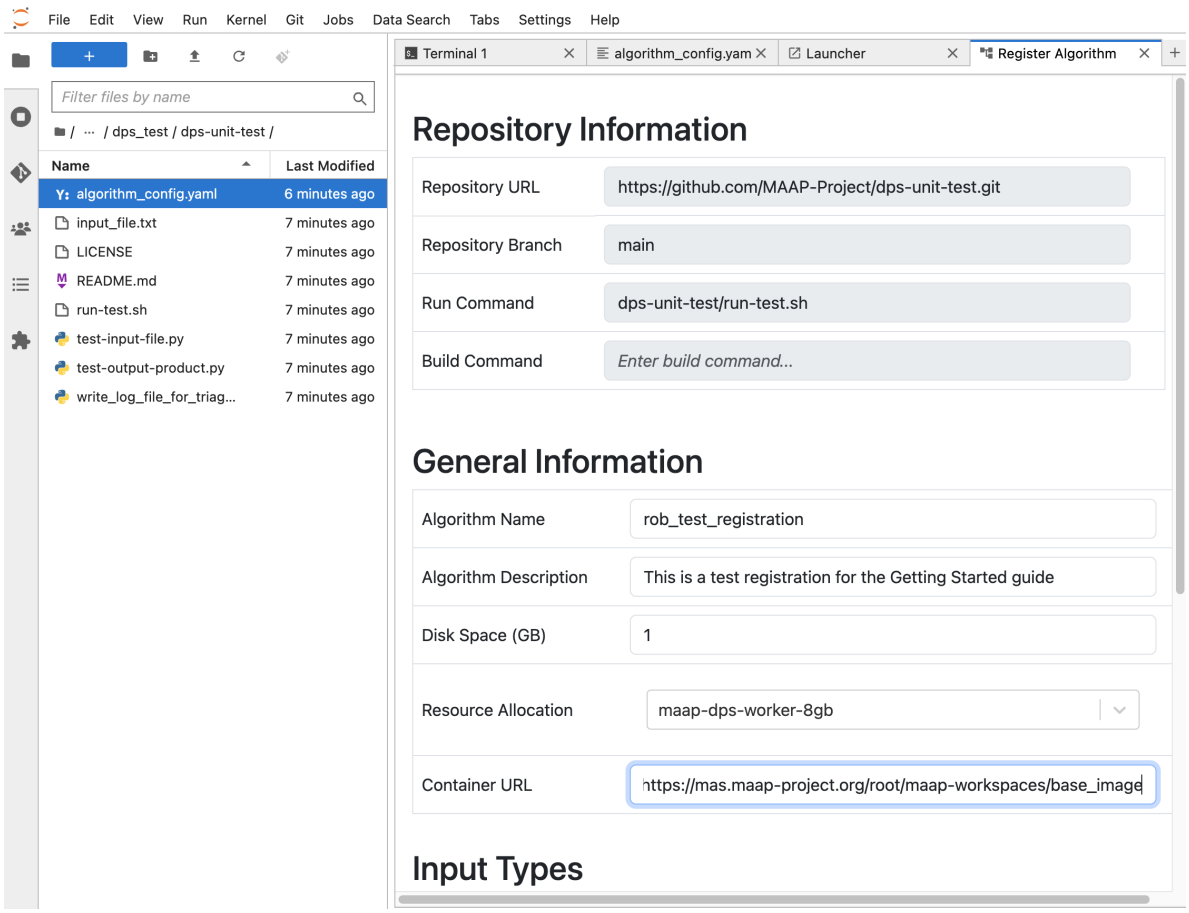
2. First you fill in the public code-repository information.

- The Repository URL is the .git URL.
- Repository Branch is used as a version when this algorithm is registered.
- The Run and Build Commands must be the full path of the scripts that will be used by the DPS to build and execute the algorithm. Typically these will be the repository_name/script_name.sh, as demonstrated in this screenshot:



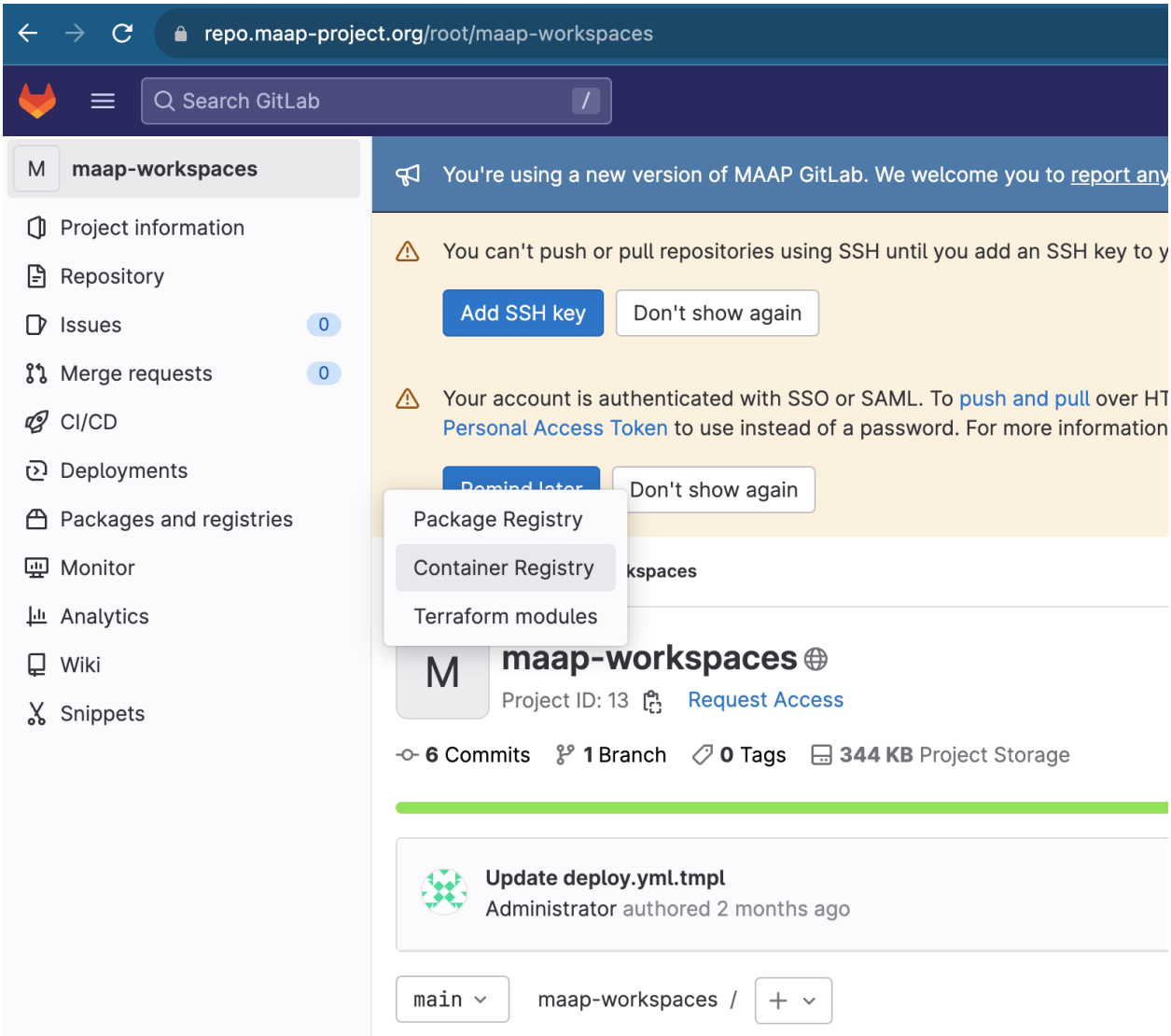
3. Once that is complete “Add General Information”.

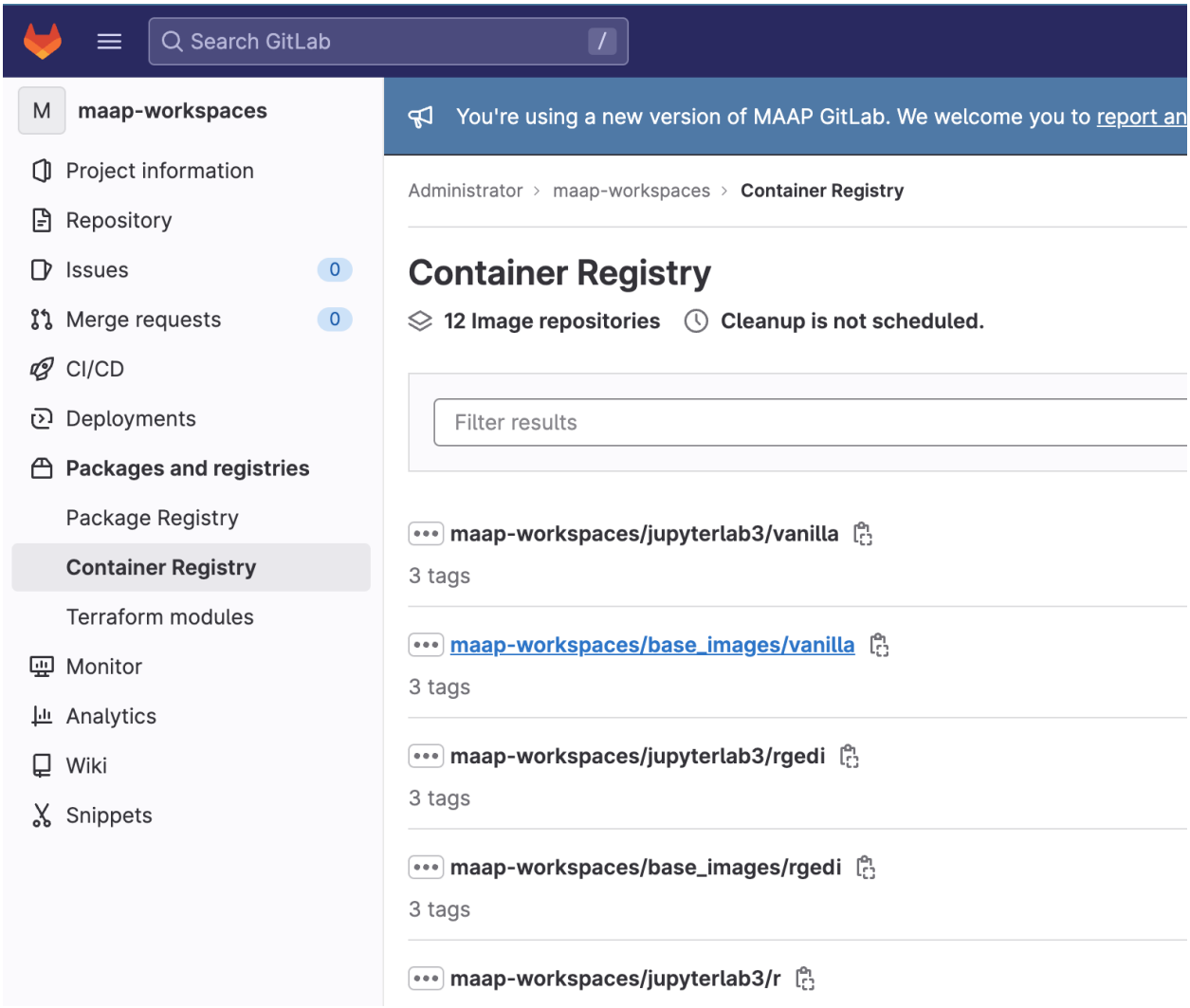
- The Algorithm Name will be the unique identifier for the algorithm in the MAAP system.
- Algorithm Description is additional free-form text to describe what this algorithm does.
- Disk Space is the minimum amount of space you expect—including all inputs, scratch, and outputs—it gives the DPS an approximation to help optimize the run.
- The Container URL is a URL of the Stack (workspace image environment) you are using as a base for the algorithm. In this example we use: `https://mas.maap-project.org/root/maap-workspaces/base_images/vanilla:v3.0.1`

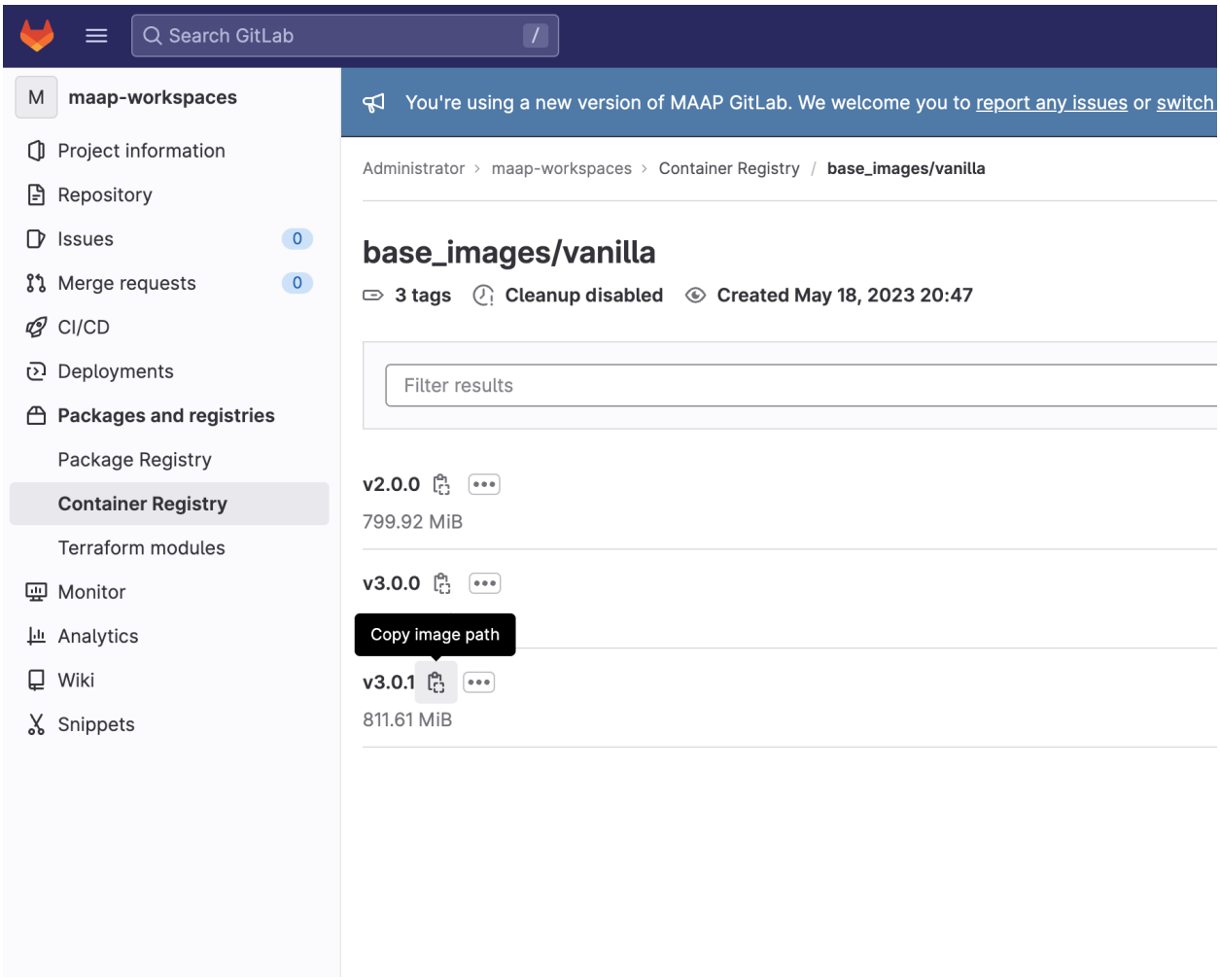


Container URLs

To find another Container URL, go to: https://repo.maap-project.org/root/maap-workspaces/container_registry (choose `Packages and Registries` > `Container registry` if you go to the main maap-workspaces area). Find your base Stack and dig in until you can copy the link of the specific version of Stack that you need, as demonstrated in these screenshots:

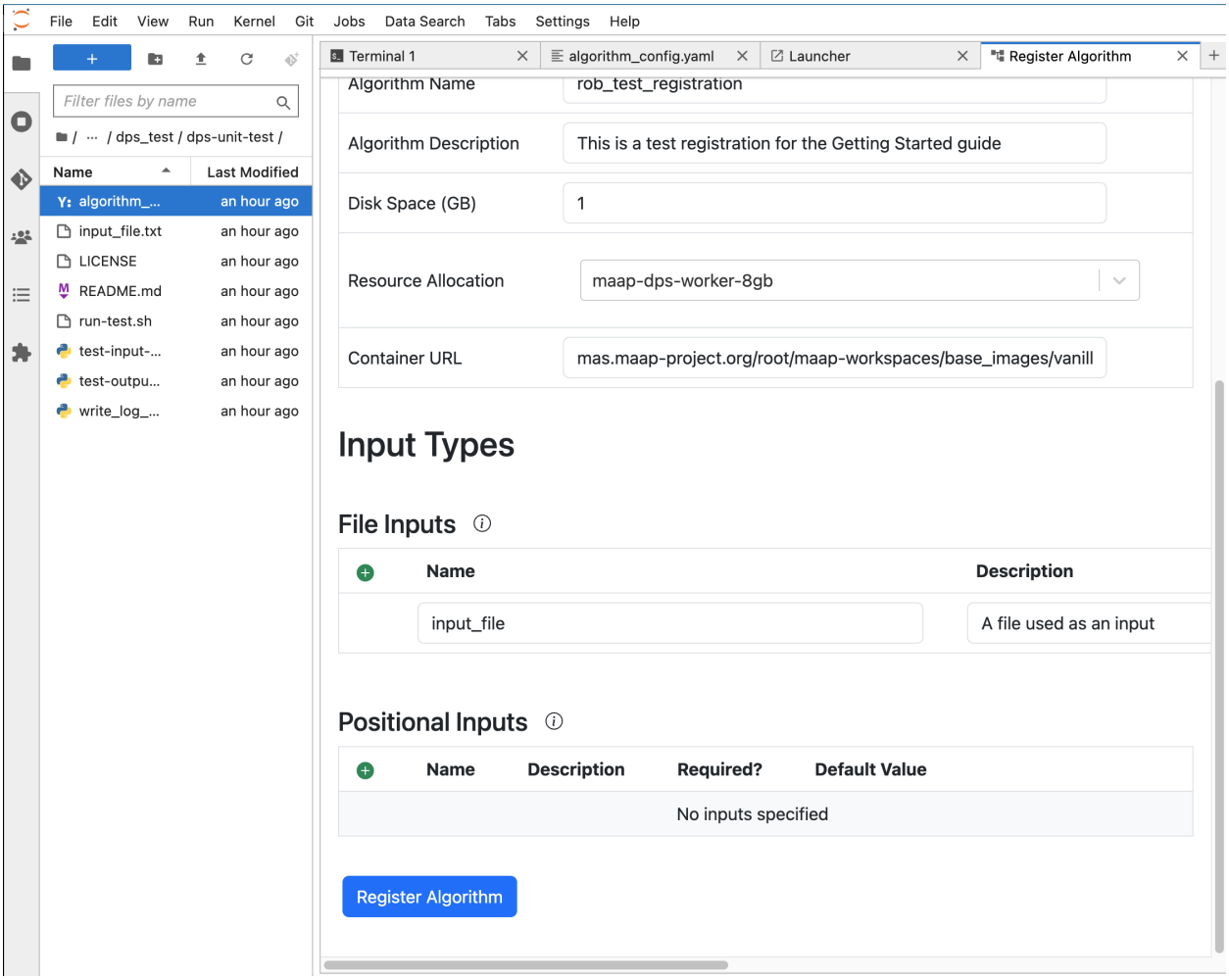

repo.maap-project.org/root/maap-workspaces
Search GitLab
/
M
maap-workspaces
Project information
Repository
Issues
0
Merge requests
0
CI/CD
Deployments
Packages and registries
Monitor
Analytics
Wiki
Snippets
You're using a new version of MAAP GitLab. We welcome you to report any
You can't push or pull repositories using SSH until you add an SSH key to y
Add SSH key
Don't show again
Your account is authenticated with SSO or SAML. To push and pull over HT
Personal Access Token to use instead of a password. For more information
Don't show again
Package Registry
Container Registry
Terraform modules
kspaces
M
maap-workspaces
Project ID: 13
Request Access
6 Commits
1 Branch
0 Tags
344 KB Project Storage
Update deploy.yml.tmpl
Administrator authored 2 months ago
main
maap-workspaces /


Search GitLab
/
M
maap-workspaces
Project information
Repository
Issues
0
Merge requests
0
CI/CD
Deployments
Packages and registries
Package Registry
Container Registry
Terraform modules
Monitor
Analytics
Wiki
Snippets
You're using a new version of MAAP GitLab. We welcome you to report an
Administrator
maap-workspaces
Container Registry
Container Registry
12 Image repositories
Cleanup is not scheduled.
Filter results
maap-workspaces/jupyterlab3/vanilla
3 tags
maap-workspaces/base_images/vanilla
3 tags
maap-workspaces/jupyterlab3/rgedi
3 tags
maap-workspaces/base_images/rgedi
3 tags
maap-workspaces/jupyterlab3/r

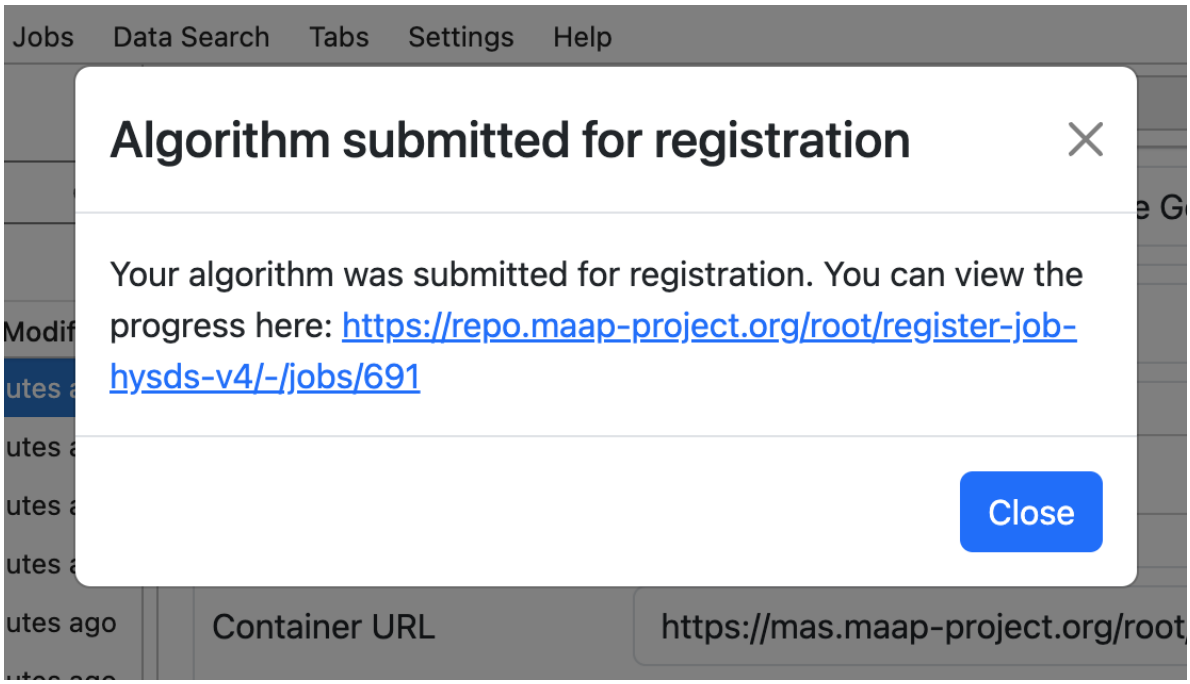


4. Fill in the Input section. There are File Inputs and Positional Inputs (command-line parameters to adjust how the algorithm runs). In our example we have on File Input called `input_file`. For each input you can add a Description, a Default Value, and mark whether it's required or optional.

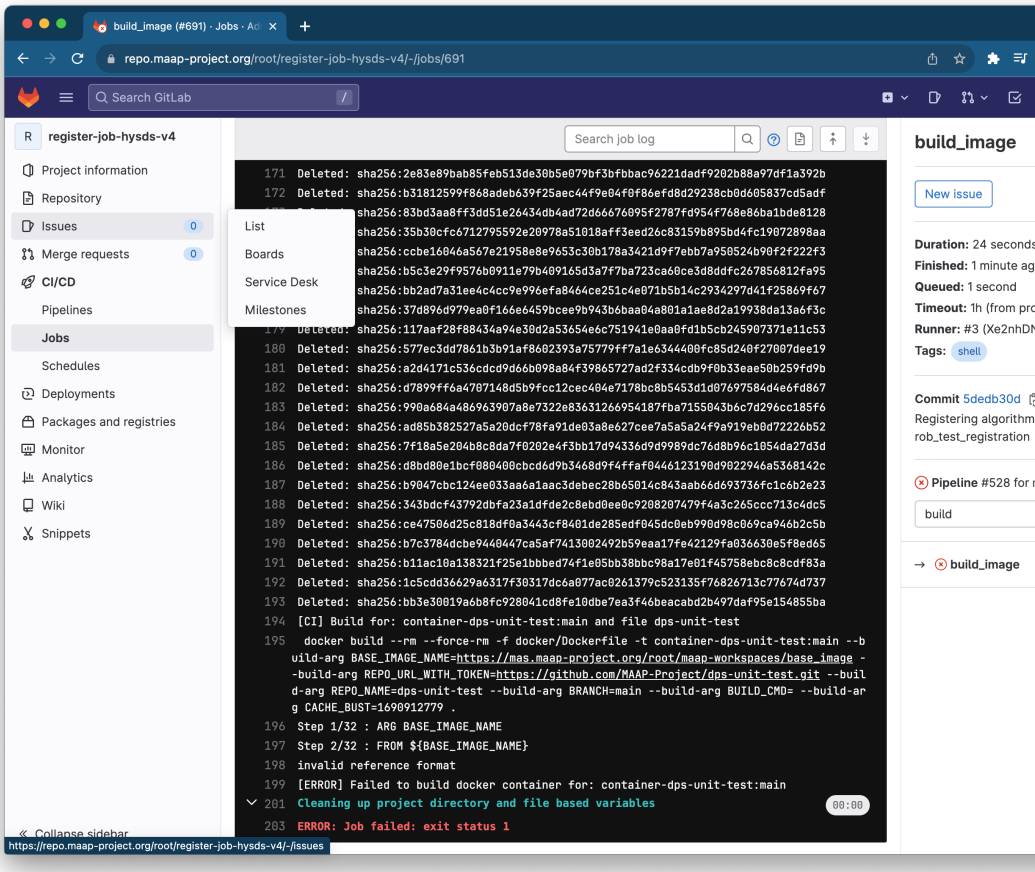
Input files are copied into `/inputs` in the working directory of your job.



5. When it looks good, press Register Algorithm at the bottom of the page. A few seconds later you should see a modal dialog with a link to the algorithm registration process.

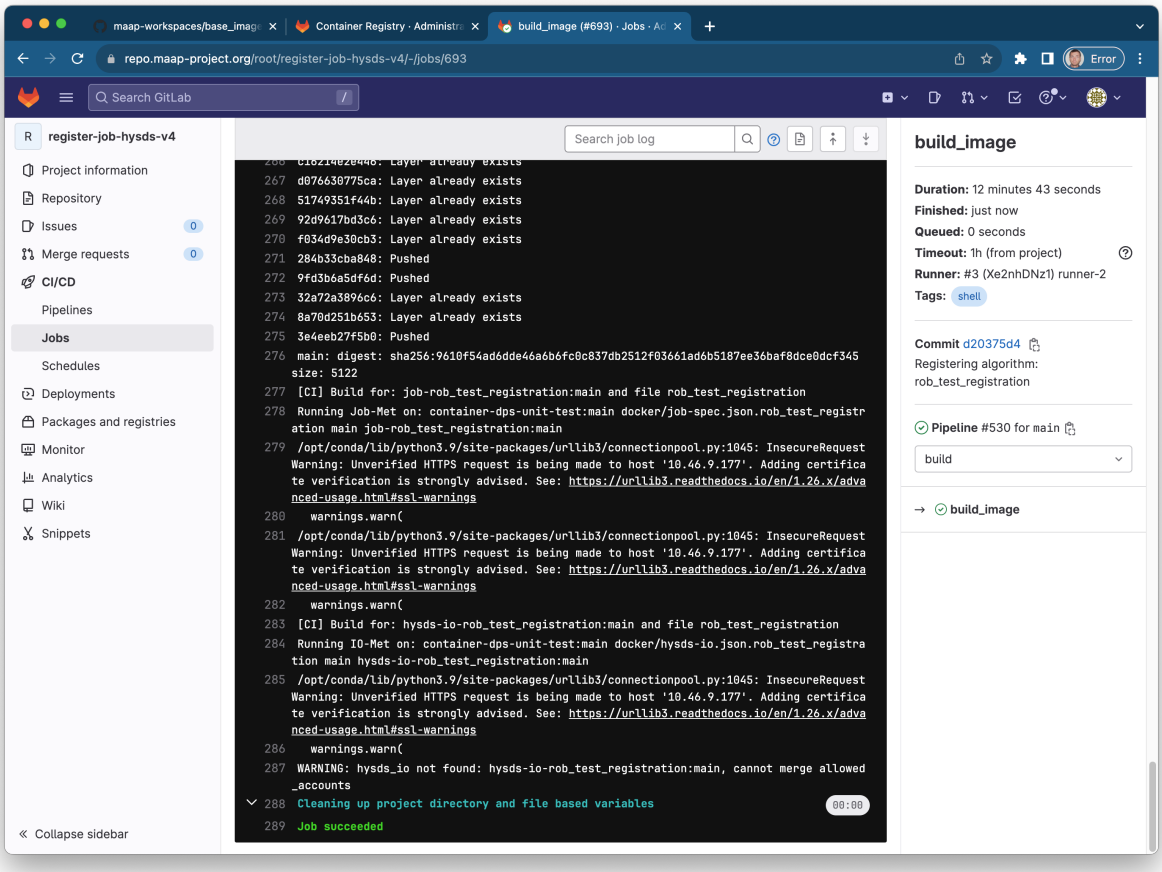


6. If you open that link in a new page or tab, you can monitor the progress of registration and see any error messages. By opening it in a new tab/window you can keep the Register Algorithm tool open and re-submit with the same values to correct any errors.

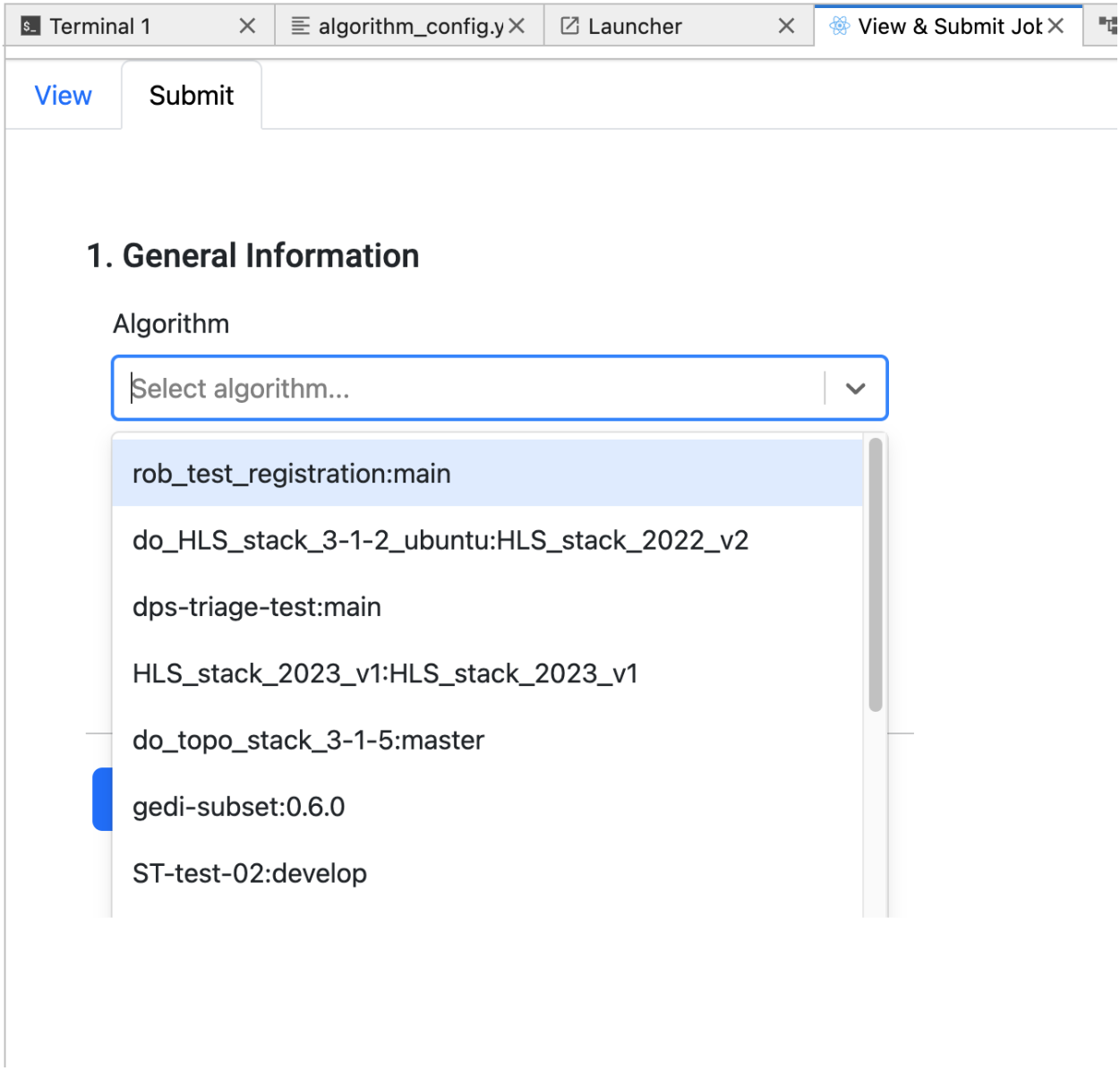


Here is an example error message:

If the process continues without failing (this may take some time) you will ultimately see "Job succeeded":



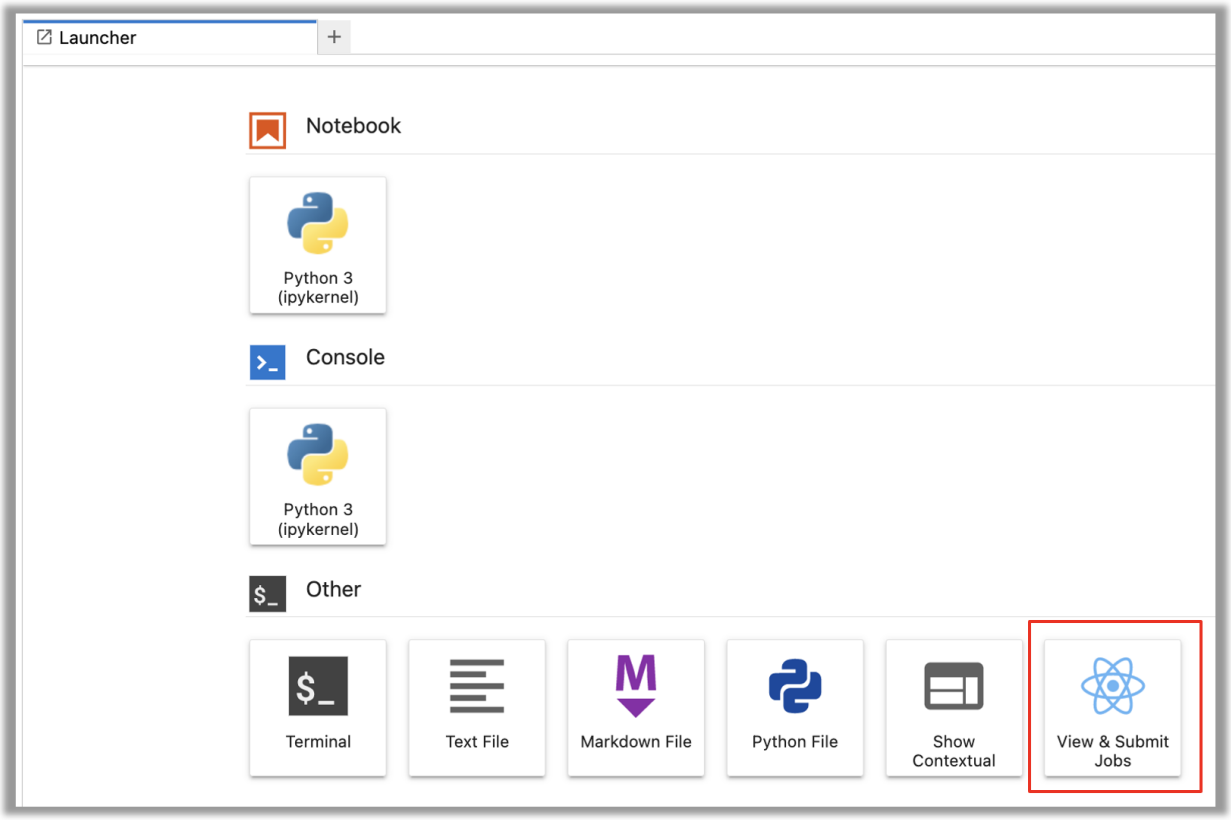
7. Now that the algorithm has registered, you will see it in the View & Submit Jobs tool in the Algorithm drop-down menu. It will be labeled with the name you put into the Algorithm Name field in the registration form you just submitted (in this example, `rob_test_registration` with version/branch `main`)



1.5.2 Run the Algorithm as a Job and Monitor it

MAAP is configured to run up to 4,000 concurrent jobs. There are two additional ways to run a job: via the Jobs UI in the Launcher, or via a call to the maap-py Python library.

The Jobs UI will let you run and monitor jobs easily. You can find full documentation in the system reference guide for *the Jobs UI* or *using maap-py* with Python in the System Reference Guide FAQs.



Some alternative methods of running the job are found below.

- Click DPS/MAS Operations menu -> Execute DPS Job
- Select your algorithm from the dropdown
- A new popup will ask for inputs; if it doesn't take inputs, the popup will say so.
- Click OK again to view the ID for the job just submitted.

OR

Import the `maap-py` library: if in Jupyter, click the small blue MAAP button in the top left corner to automatically insert code. If using a script, add these lines manually at the top of your notebook:

```
from maap.maap import MAAP
maap = MAAP()
```

Pass your algorithm's name, version, required inputs, and username to the function `maap.submitJob (identifier is job- algo_name:algo_version)` Check result: `maap.getJobResult()`

CHAPTER

TWO

SCIENCE EXAMPLES

2.1 HLSL30 Search and Composite

Authors: Nathan Thomas (GSFC/UMD), Sumant Jha (MSFC/USRA), Aimee Barciauskas (DevSeed), Alex Mandel (DevSeed)

Date: December 19, 2022

Description: In this tutorial, we will search the LPDAAC for Harmonized Landsat Sentinel-2 (HLS) 30m optical imagery that intersects an AOI. We will filter the catalog based on a cloud cover % and build a maximum-NDVI (Normalized Difference Vegetation Index) composite image, including a suite of popular indices, which will help give us an in-depth look at vegetation health.

2.1.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.1.2 About the Data

Harmonized Landsat Sentinel-2 30m

Harmonized Landsat Sentinel-2 (HLS) was developed in response to a greater need for moderate-to-high resolution imagery to track various short-term landcover changes. Data are gathered by the Landsat-8 and Landsat-9 satellites, which carry the Operational Land Imager (OLI), as well as the Sentinel-2A and Sentinel-2B satellites, which carry the Multi-Spectral Instrument (MSI). With combined measurements from both the Landsat and Sentinel satellites, HLS imagery has global coverage with a spatial resolution of 30m and a temporal resolution of 2-3 days. (Source: HLS Overview Page)

Note about HLS datasets: The Sentinel and Landsat assets have been "harmonized" in the sense that these products have been generated to use the same spatial resolution and grid system. Thus, the HLS S30 and L30 products can be used interchangeably in algorithms and are "stackable". However, the individual band assets are specific to each provider.

2.1.3 Additional Resources

- HLSL30 v002 Dataset Landing Page
- Landsat 8 Bands and Combinations Blog
- HLSS30 v002 Dataset Landing Page
- Sentinel 2 Bands and Combinations Blog
- Harmonized Landsat Sentinel-2 (HLS) User Guide

2.1.4 Importing and Installing Packages

We will begin by installing any packages we need and importing the packages that we will use.

If needed the following packages should be installed:

[1]:
```
# cleanup data: removes data files that should be replaced
!rm -rf ./local-s3.json
!rm -rf ./sample.json

if False:
    !conda install -c conda-forge pystac-client -y
    !conda install -c conda-forge rio-tiler -y
```

Prerequisites

- geopandas
- folium
- pystac-client
- rio_tiler

[ ]:
```
# Uncomment the following lines to install these packages if you haven't already.
# !pip install geopandas
# !pip install folium
# !pip install pystac-client
# !pip install rio_tiler
```

We will now import a suite of packages that we will need:

[3]:
```
from maap.maap import MAAP
maap = MAAP(maap_host='api.maap-project.org')
import geopandas as gpd
import folium
import h5py
import pandas
import matplotlib
import matplotlib.pyplot as plt
from shapely.geometry import Polygon
from pystac_client import Client
import datetime
import os
import rasterio as rio
```

(continues on next page)

(continued from previous page)

```
import boto3
import json
import botocore
from rasterio.session import AWSSession
from rio_tiler.io import COGReader
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
```

```
/opt/conda/lib/python3.7/site-packages/geopandas/_compat.py:115: UserWarning: The␣
↪Shapely GEOS version (3.11.1-CAPI-1.17.1) is incompatible with the GEOS version PyGEOS␣
↪was compiled with (3.8.1-CAPI-1.13.3). Conversions between both will be slow.
  shapely_geos_version, geos_capi_version_string
```

2.1.5 Creating an AOI

To begin we will create a polygon in a forested area in Virginia, USA. We will do this by providing a number of lat/lon coordinates and creating an AOI.

[4]:
```
lon_coords = [-80, -80, -79., -79, -80]
lat_coords = [39, 38.2, 38.2, 39, 39]

polygon_geom = Polygon(zip(lon_coords, lat_coords))
crs = 'epsg:4326'
AOI = gpd.GeoDataFrame(index=[0], crs=crs, geometry=[polygon_geom])
AOI_bbox = AOI.bounds.iloc[0].to_list()
```

We can visualize this polygon using a folium interactive map.

[5]:
```
m = folium.Map([38.5,-79.3], zoom_start=9, tiles='OpenStreetMap')
folium.GeoJson(AOI).add_to(m)
folium.LatLngPopup().add_to(m)
m
```

[5]:
```
<folium.folium.Map at 0x7f223c58c090>
```

2.1.6 Accessing the HLS Data

To be able to access the HLS imagery we need to activate a 'rasterio' AWS session. This will give us the required access keys that we need to search and read data from the LPDAAC S3 bucket.

[6]:
```
def get_aws_session_DAAC():
    """Create a Rasterio AWS Session with Credentials"""
    creds = maap.aws.earthdata_s3_credentials('https://data.lpdaac.earthdatacloud.nasa.
↪gov/s3credentials')
    boto3_session = boto3.Session(
        aws_access_key_id=creds['accessKeyId'],
        aws_secret_access_key=creds['secretAccessKey'],
        aws_session_token=creds['sessionToken'],
        region_name='us-west-2'
```

(continues on next page)

(continued from previous page)

```
    )
    return AWSSession(boto3_session)
```

[7]:
```
print('Getting AWS credentials...')
aws_session = get_aws_session_DAAC()
print('Finished')
```

```
Getting AWS credentials...
Finished
```

Now that we have our session credentials set up, we can search the HLS catalog using the following function, filtering by spatial extent (our AOI) and a time window:

[8]:
```
def query_stac(year, bbox, max_cloud, api, start_month_day, end_month_day):
    print('opening client')
    catalog = Client.open(api)

    date_min = str(year) + '-' + start_month_day
    print('start_month_day:\t\t', start_month_day)
    print('end_month_day:\t\t', end_month_day)
    date_max = str(year) + '-' + end_month_day
    start_date = datetime.datetime.strptime(date_min, "%Y-%m-%d")
    end_date = datetime.datetime.strptime(date_max, "%Y-%m-%d")
    start = start_date.strftime("%Y-%m-%dT00:00:00Z")
    end = end_date.strftime("%Y-%m-%dT23:59:59Z")

    print('start date, end date:\t\t', start, end)

    print('\nConducting HLS search now...')

    search = catalog.search(
        collections=["HLSL30.v2.0"],
        datetime=[start,end],
        bbox=bbox,
        max_items=500, # for testing, and keep it from hanging
        # query={"eo:cloud_cover":{"lt":20}} #doesn't work
    )
    print(f"Search query parameters:\n{search}\n")
    results = search.get_all_items_as_dict()

    return results
```

Here we run our STAC search and write the results out to a JSON file so we can access it later.

[9]:
```
search = query_stac(2020, AOI_bbox, 20, 'https://cmr.earthdata.nasa.gov/stac/LPCLOUD',
↪'06-01', '09-30')

with open("./sample.json", "w") as outfile:
    json.dump(search, outfile)
```

```
opening client
start_month_day:                06-01
end_month_day:                  09-30
```

(continues on next page)

(continued from previous page)

```
start date, end date:            2020-06-01T00:00:00Z 2020-09-30T23:59:59Z

Conducting HLS search now...
Search query parameters:
<pystac_client.item_search.ItemSearch object at 0x7f223c514850>
```

So far, we have not filtered by cloud cover, which is a common filtering parameter for optical imagery. We can use the metadata files included in our STAC search to find the amount of cloud cover in each file and decide if it meets our threshold or not. We will set a cloud cover threshold of 50%. While we are doing this, we will also change the URLs to access the optical imagery from "https://" to AWS S3 URLs ("S3://").

[10]:
```
def write_local_data_and_catalog_s3(catalog, bands, save_path, cloud_cover, s3_path="s3:/
↪/"):
    "Given path to a response json from a sat-api query, make a copy changing urls to␣
↪local paths"
    creds = maap.aws.earthdata_s3_credentials('https://data.lpdaac.earthdatacloud.nasa.
↪gov/s3credentials')
    aws_session = boto3.session.Session(
        aws_access_key_id=creds['accessKeyId'],
        aws_secret_access_key=creds['secretAccessKey'],
        aws_session_token=creds['sessionToken'],
        region_name='us-west-2')
    s3 = aws_session.client('s3')

    with open(catalog) as f:
        clean_features = []
        asset_catalog = json.load(f)

        # Remove duplicate scenes
        features = asset_catalog['features']

        for feature in features:
            umm_json = feature['links'][6]['href']
            umm_data = !curl -i {umm_json}
            for line in umm_data:
                if "CLOUD_COVERAGE" in line:
                    cc_perc = (int(line.split("CLOUD_COVERAGE")[-1].split('"')[4]))
                    if cc_perc > cloud_cover:
                        pass
                    else:
                        try:
                            for band in bands:
                                output_file = feature['assets'][band]['href'].replace(
↪'https://data.lpdaac.earthdatacloud.nasa.gov/', s3_path)
                                bucket_name = output_file.split("/")[2]
                                s3_key = "/".join(output_file.split("/")[3:])
                                head = s3.head_object(Bucket = bucket_name, Key = s3_key,
↪ RequestPayer='requester')
                                if head['ResponseMetadata']['HTTPStatusCode'] == 200:
                                    feature['assets'][band]['href'] = output_file
                            clean_features.append(feature)
```

(continues on next page)

(continued from previous page)

```
                        except botocore.exceptions.ClientError as e:
                            if e.response['Error']['Code'] == "404":
                                print(f"The object does not exist. {output_file}")
                            else:
                                raise
        # save and updated catalog with local paths
        asset_catalog['features'] = clean_features
        local_catalog = catalog.replace('sample', 'local-s3')
        with open(local_catalog,'w') as jsonfile:
            json.dump(asset_catalog, jsonfile)

        return local_catalog
```

When run, this will create a new JSON file that will only include files that meet our cloud cover threshold and have S3 URLs.

[11]:
```
local_cat = write_local_data_and_catalog_s3('./sample.json', ['B02','B03','B04','B05',
↪'B06','B07','Fmask'], './', 60, s3_path="s3://")
```

Now that we have images that meet our requirements, we will composite them into a multiband image for our AOI. We will composite the image on a band-by-band basis, so we first have to get a list of all the file paths for each band.

[12]:
```
def GetBandLists(inJSON, bandnum, tiles=['']):
    bands = dict({2:'B02', 3:'B03', 4:'B04', 5:'B05', 6:'B06', 7:'B07',8:'Fmask'})
    BandList = []
    with open(inJSON) as f:
        response = json.load(f)
    for i in range(len(response['features'])):
        try:
            getBand = response['features'][i]['assets'][bands[bandnum]]['href']
            # check 's3' is at position [:2]
            if getBand.startswith('s3', 0, 2):
                BandList.append(getBand)
        except Exception as e:
            print(e)

    BandList.sort()
    return BandList
```

We will build a band list of file paths for each image band. We will also access the 'fmask' band to mask out clouds.

[13]:
```
blue_bands = GetBandLists('./local-s3.json', 2)
green_bands = GetBandLists('./local-s3.json', 3)
red_bands = GetBandLists('./local-s3.json', 4)
nir_bands = GetBandLists('./local-s3.json', 5)
swir_bands = GetBandLists('./local-s3.json', 6)
swir2_bands = GetBandLists('./local-s3.json', 7)
fmask_bands = GetBandLists('./local-s3.json', 8)

print('number of each images in each band = ', len(blue_bands))
```

```
number of each images in each band =  21
```

2.1.7 Reading in HLS Data and Creating Composite

We will not read all of the HLS data, as we only need what's included in our AOI. To do this "windowed read" we need to know the dimensions, in pixels, of the window. To do this we need to convert our AOI to a projected coordinate system (UTM) and calculate the number of columns and rows we will need, using a pixel resolution of 30m.

[14]:
```
def get_shape(bbox, res=30):
    left, bottom, right, top = bbox
    width = int((right-left)/res)
    height = int((top-bottom)/res)

    return height,width

# convert to m
AOI_utm = AOI.to_crs('epsg:32617')
height, width = get_shape(AOI_utm.bounds.iloc[0].to_list())
```

When building a maximum-NDVI composite, the first data we need is the red and NIR bands to make an NDVI band for each image. We will use 'riotiler' to perform a windowed read of our data. We will also read the 'fmask' layer as we can use this to mask out unwanted pixels.

[15]:
```
def ReadData(file, in_bbox, height, width, epsg="epsg:4326", dst_crs="epsg:4326"):
    '''Read a window of data from the raster matching the tile bbox'''
    with COGReader(file) as cog:
        img = cog.part(in_bbox, bounds_crs=epsg, max_size=None, dst_crs=dst_crs,␣
↪height=height, width=width)

    return (np.squeeze(img.as_masked().astype(np.float32)) * 0.0001)
```

Our AWS session has an expiry time. Now would be a good time to renew our access key to ensure we do not encounter any timeout issues.

[16]:
```
print('Getting AWS credentials...')
aws_session = get_aws_session_DAAC()
print('Finished')
```

```
Getting AWS credentials...
Finished
```

Using our renewed session, for each band we will read in the relevant band in each of our images, creating an array of image bands.

[17]:
```
with rio.Env(aws_session):
    print('reading red bands...')
    red_stack = np.array([ReadData(red_bands[i], AOI_bbox, height, width, epsg="epsg:4326
↪", dst_crs="epsg:4326") for i in range(len(red_bands))])
    print('reading nir bands...')
    nir_stack = np.array([ReadData(nir_bands[i], AOI_bbox, height, width, epsg="epsg:4326
↪", dst_crs="epsg:4326") for i in range(len(nir_bands))])
    print('reading fmask bands...')
    fmask_stack = np.array([ReadData(fmask_bands[i], AOI_bbox, height, width, epsg="epsg:
↪4326", dst_crs="epsg:4326") for i in range(len(fmask_bands))])
    print('finished')
```

(continues on next page)

(continued from previous page)

```
print("number of red_bands = ", np.shape(red_stack)[0])
```

```
reading red bands...
reading nir bands...
reading fmask bands...
finished
number of red_bands =  21
```

Now that we have our red band array and NIR band array we can make an NDVI image. While we do this, we will also apply our 'fmask' to remove any pixels that contain clouds.

[18]:
```
ndvi_stack =  np.ma.array(np.where(((fmask_stack==1)), -9999, (nir_stack-red_stack)/(nir_
↪stack+red_stack)))
ndvi_stack = np.where((~np.isfinite(ndvi_stack)) | (ndvi_stack>1), -9999, ndvi_stack)
```

```
/opt/conda/lib/python3.7/site-packages/ipykernel_launcher.py:1: RuntimeWarning: divide␣
↪by zero encountered in true_divide
  """Entry point for launching an IPython kernel.
```

At this point, we can plot our images and see which ones contain cloud coverage. These images have gaps where there is no data or cloud coverage.

[19]:
```
fig, axes = plt.subplots(3,7, figsize=(33,30))

for i, ax in enumerate(axes.flat):
    ndvi_stack[i] = np.where((ndvi_stack[i]>1) | (ndvi_stack[i]<-1), 0, ndvi_stack[i])
    ax.axes.xaxis.set_visible(False)
    ax.axes.yaxis.set_visible(False)
    ax.imshow(ndvi_stack[i], cmap='viridis', clim=(0.1, 0.5))
```

Now that we have a stack of NDVI bands, we can create an index band that maps the pixel position from each band where the NDVI value is greatest. We can use this to locate the pixels we want to use in our composite.

[20]:
```
# Create Bool mask where there is no value in any of the NDVI layers
print("Make NDVI valid mask")
print("shape:\t\t", np.ma.array(ndvi_stack).shape)
MaxNDVI = np.ma.max(np.ma.array(ndvi_stack),axis=0)
BoolMask = np.ma.getmask(MaxNDVI)
del MaxNDVI

## Get the argmax index positions from the stack of NDVI images
print("Get stack nan mask")
ndvi_stack = np.ma.array(ndvi_stack)
print("Calculate Stack max NDVI image")
NDVImax = np.nanargmax(ndvi_stack,axis=0)
## create a tmp array (binary mask) of the same input shape
NDVItmp = np.ma.zeros(ndvi_stack.shape, dtype=bool)

## for each dimension assign the index position (flattens the array to a LUT)
print("Create LUT of max NDVI positions")
for i in range(np.shape(ndvi_stack)[0]):
    NDVItmp[i,:,:]=NDVImax==i
```

```
Make NDVI valid mask
shape:           (21, 3006, 2951)
Get stack nan mask
Calculate Stack max NDVI image
Create LUT of max NDVI positions
```

Now that we have our NDVI lookup table and a list of all the images for each band, we can make composites based on the maximum NDVI value. For this, we will use the following two functions:

[21]:
```
def CollapseBands(inArr, NDVItmp, BoolMask):
    inArr = np.ma.masked_equal(inArr, 0)
    inArr[np.logical_not(NDVItmp)]=0
    compImg = np.ma.masked_array(inArr.sum(0), BoolMask)
    #print(compImg)
    return compImg

def CreateComposite(file_list, NDVItmp, BoolMask, in_bbox, height, width, epsg, dst_crs):
    MaskedFile = [ReadData(file_list[i], in_bbox, height, width, epsg, dst_crs) for i in␣
↪range(len(file_list))]
    Composite=CollapseBands(MaskedFile, NDVItmp, BoolMask)
    return Composite
```

For each band, we will read all the images (for the required band) and create a composite based on the maximum NDVI value.

[22]:
```
aws_session = get_aws_session_DAAC()
with rio.Env(aws_session):
    print('Creating Blue Composite')
    blue_comp = CreateComposite(blue_bands, NDVItmp, BoolMask, AOI_bbox, height, width,
↪"epsg:4326", "epsg:4326")
    print('Creating Green Composite')
    green_comp = CreateComposite(green_bands, NDVItmp, BoolMask, AOI_bbox, height, width,
↪ "epsg:4326", "epsg:4326")
    print('Creating Red Composite')
    red_comp = CreateComposite(red_bands, NDVItmp, BoolMask, AOI_bbox, height, width,
↪"epsg:4326", "epsg:4326")
    print('Creating NIR Composite')
    nir_comp = CreateComposite(nir_bands, NDVItmp, BoolMask, AOI_bbox, height, width,
↪"epsg:4326", "epsg:4326")
    print('Creating SWIR Composite')
    swir_comp = CreateComposite(swir_bands, NDVItmp, BoolMask, AOI_bbox, height, width,
↪"epsg:4326", "epsg:4326")
    print('Creating SWIR2 Composite')
    swir2_comp = CreateComposite(swir2_bands, NDVItmp, BoolMask, AOI_bbox, height, width,
↪ "epsg:4326", "epsg:4326")
    print('Creating NDVI Composite')
    ndvi_comp = CollapseBands(ndvi_stack, NDVItmp, BoolMask)
    print('Creating fmask Composite')
    fmask_comp = CollapseBands(fmask_stack, NDVItmp, BoolMask)
```

```
Creating Blue Composite
Creating Green Composite
Creating Red Composite
```

(continues on next page)

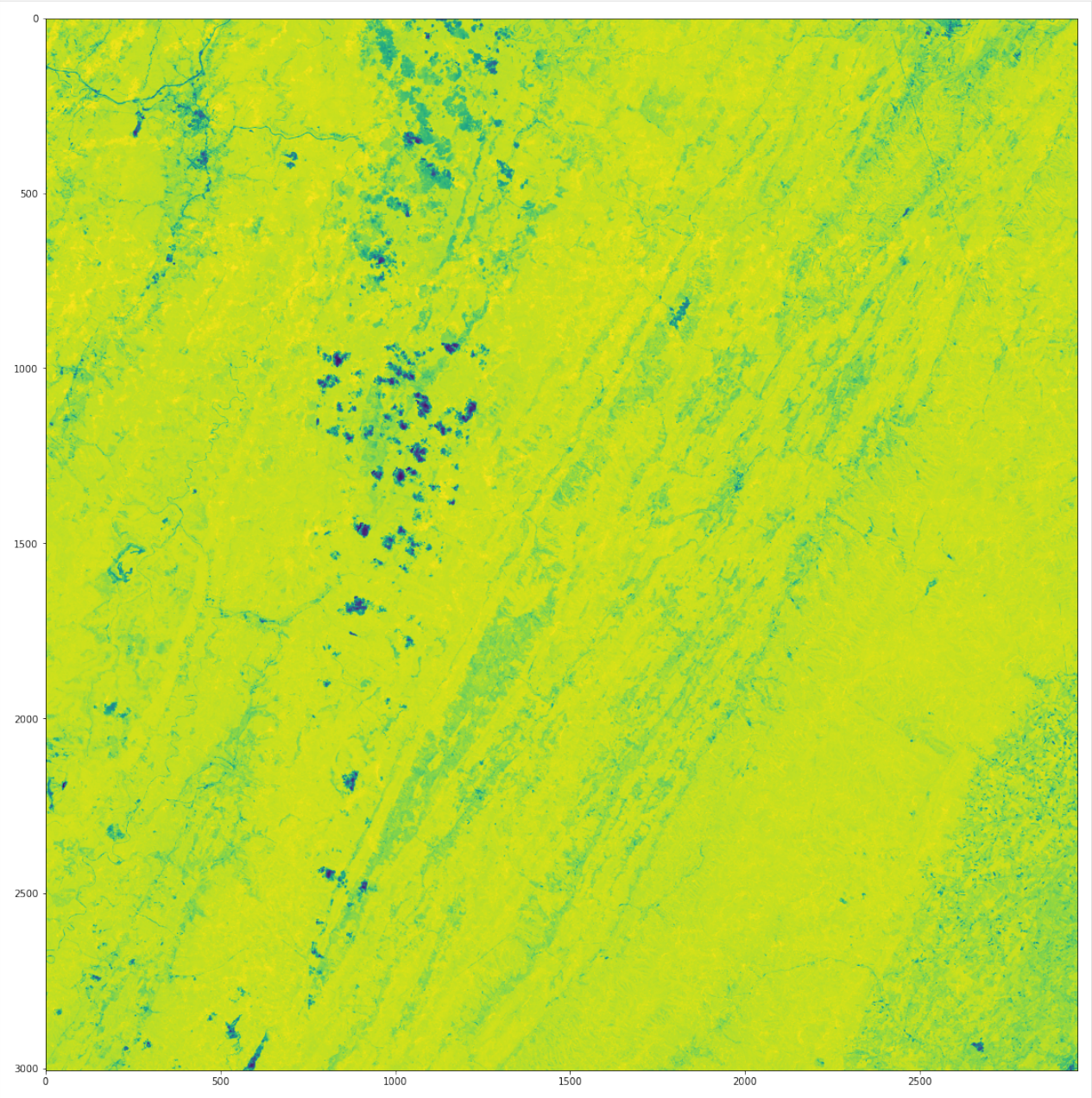
(continued from previous page)

```
Creating NIR Composite
Creating SWIR Composite
Creating SWIR2 Composite
Creating NDVI Composite
Creating fmask Composite
```

We can look at our NDVI composite image and see we now have a complete image for our AOI.

[23]:
```
fig, axes = plt.subplots(1,1, figsize=(20,20))
ax.axes.xaxis.set_visible(False)
ax.axes.yaxis.set_visible(False)
plt.imshow(np.where(fmask_comp==1, -9999, ndvi_comp))
```

[23]:
```
<matplotlib.image.AxesImage at 0x7f223c43f6d0>
```



Now that we have a 7-band composite image, we can use these bands to calculate a suite of common vegetation indices using the following functions. These indices will give us a better look at vegetation health by giving us information on vegetation water content, greenness, and more.

[24]:
```
# SAVI
def calcSAVI(red, nir):
    savi = ((nir - red)/(nir + red + 0.5))*(1.5)
    print('\tSAVI Created')
    return savi

# NDMI
def calcNDMI(nir, swir):
```

(continues on next page)

(continued from previous page)

```
    ndmi = (nir - swir)/(nir + swir)
    print('\tNDMI Created')
    return ndmi

# EVI
def calcEVI(blue, red, nir):
    evi = 2.5 * ((nir - red) / (nir + 6 * red - 7.5 * blue + 1))
    print('\tEVI Created')
    return evi

# NBR
def calcNBR(nir, swir2):
    nbr = (nir - swir2)/(nir + swir2)
    print('\tNBR Created')
    return nbr

# MSAVI
def calcMSAVI(red, nir):
    msavi = (2 * nir + 1 - np.sqrt((2 * nir + 1)**2 - 8 * (nir - red))) / 2
    print('\tMSAVI Created')
    return msavi
```

We can call these functions and make our additional indices.

[25]:
```
# calculate covars
print("Generating covariates")
SAVI = calcSAVI(red_comp, nir_comp)
#print("NDMI")
NDMI = calcNDMI(nir_comp, swir_comp)
#print("EVI")
EVI = calcEVI(blue_comp, red_comp, nir_comp)
#print("NBR")
NBR = calcNBR(nir_comp, swir2_comp)
MSAVI = calcMSAVI(red_comp, nir_comp)
```

```
Generating covariates
        SAVI Created
        NDMI Created
        EVI Created
        NBR Created
        MSAVI Created
```

We have a suite of 12 bands now, and we can merge them together into a single 12-band image stack.

[26]:
```
print("\nCreating raster stack...\n")
stack = np.transpose([blue_comp, green_comp, red_comp, nir_comp, swir_comp, swir2_comp,␣
↪ndvi_comp, SAVI, MSAVI, NDMI, EVI, NBR], [0, 1, 2])
stack = np.where(fmask_comp==1, -9999, stack)
print(np.shape(stack))
```

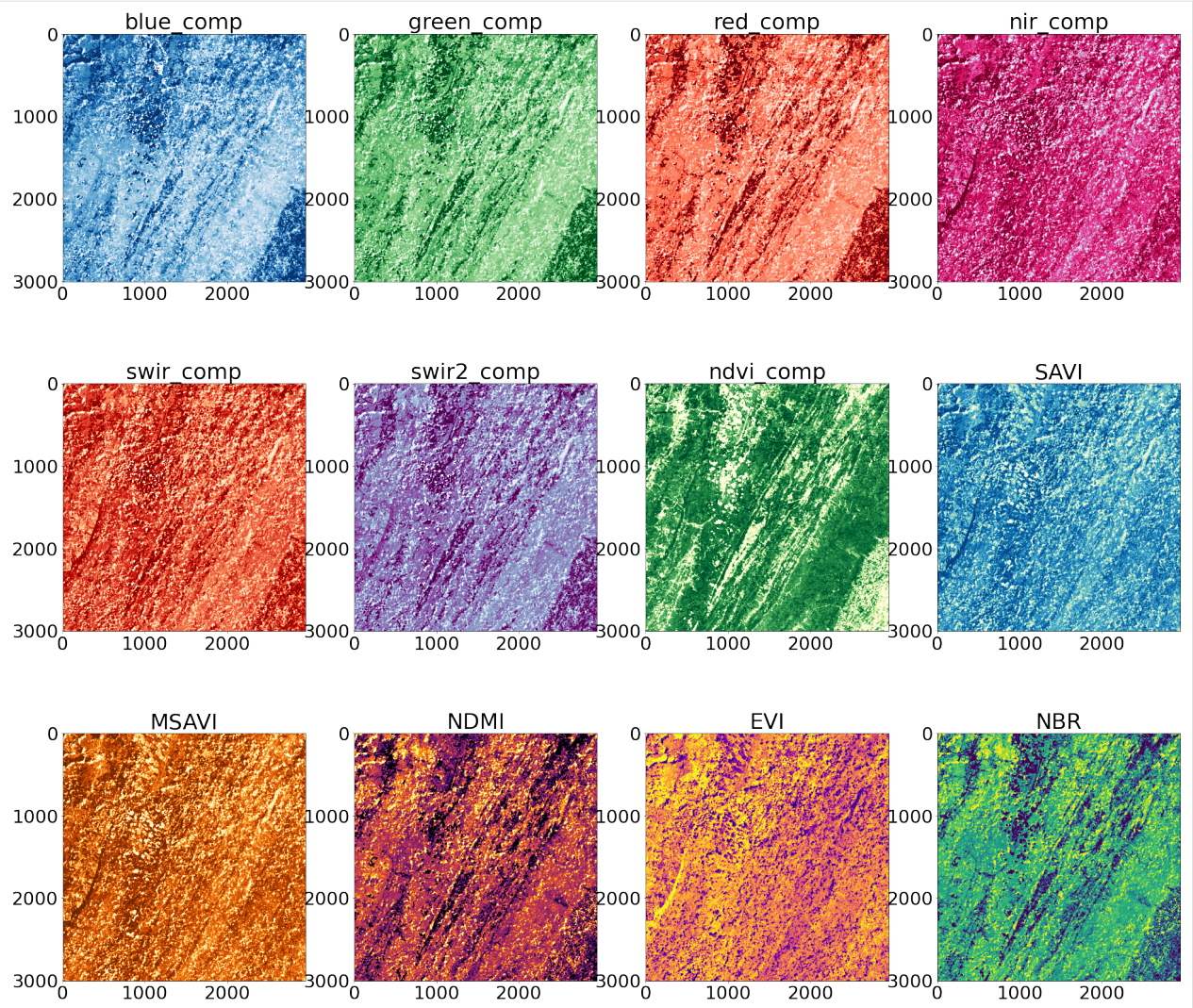
```

Creating raster stack...

(12, 3006, 2951)
```

2.1.8 Display Results

We can look at each of these bands by using 'matplotlib' to plot each one individually.

[27]:
```
n: int = len(stack)
#topo_cmaps = ["bone","Spectral", "magma", "RdBu", "coolwarm"]
topo_cmaps = ['Blues','Greens','Reds','PuRd','OrRd','BuPu','YlGn','GnBu','YlOrBr',
↪'inferno','plasma', 'viridis']
print(stack.shape)
bandnames = ['blue_comp', 'green_comp', 'red_comp', 'nir_comp', 'swir_comp', 'swir2_comp
↪', 'ndvi_comp', 'SAVI', 'MSAVI', 'NDMI', 'EVI', 'NBR']

font = {'size'   : 30}
matplotlib.rc('font', **font)

#axs = [[0,0],[0,1],[0,2],[0,3],[0,4],[0,5],[1,0],[1,1],[1,2],[1,3],[1,4],[1,5],[2,0],[2,
↪1],[2,2],[2,3],[2,4],[2,5]]
fig, axes = plt.subplots(3,4, figsize=(33,30))
print(axes.flat)
for i, ax in enumerate(axes.flat):
    ax.imshow(stack[i], cmap=topo_cmaps[i], clim=(np.percentile(stack[i], 10), np.
↪percentile(stack[i], 90)))
    ax.set_title(bandnames[i])
```

```
(12, 3006, 2951)
<numpy.flatiter object at 0x561c7d6fe260>
```



We can also visualize our composite NDVI band on our interactive 'folium' map. You can see that even though we found multiple images, by using a windowed read we were able to just read and process the data we needed.

[28]:
```
m = folium.Map(location=[38.6, -78.5], zoom_start=9, tiles='CartoDB positron')
AOI_bx = AOI.bounds
#folium.GeoJson(AOI, style_function=lambda x: {'fillColor': 'orange','opacity':0}).add_to(m)
geo_r = folium.raster_layers.ImageOverlay(np.ma.getdata(ndvi_comp), opacity=1,␣
↪bounds=[[AOI_bx['miny'][0],AOI_bx['minx'][0]],[AOI_bx['maxy'][0],AOI_bx['maxx'][0]]])
geo_r.add_to(m)

m
```

[28]:
```
<folium.folium.Map at 0x7f236144a950>
```

2.2 GEDI_L2A Search and Visualize

Authors: Nathan Thomas (GSFC/UMD), Sumant Jha (MSFC/USRA), Samuel Ayers (UAH)

Date: March 9, 2023

Description: This tutorial aims to provide information and code to help users get started working with the Global Ecosystem Dynamics Investigation (GEDI) Level 2A (GEDI_L2A) product using the MAAP. We will search for the data within NASA's Common Metadata Repository (CMR) and plot the orbit path.

2.2.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.2.2 About the Data

GEDI L2A Elevation and Height Metrics Data Global Footprint Level

This dataset provides Global Ecosystem Dynamics Investigation (GEDI) Level 2A (L2A) data, which has the purpose of providing waveform interpretation and extracted products from the GEDI L1B waveforms. These products include ground elevation, canopy top height, and relative height (RH) metrics. GEDI is attached to the International Space Station (ISS) and collects data globally between 51.6° N and 51.6° S latitudes at the highest resolution and densest sampling of any light detection and ranging (lidar) instrument in orbit to date; specifically, GEDI L2A data has a spatial resolution of 25m. (Source: GEDI L2A Dataset Landing Page)

2.2.3 Additional Resources

- Earthdata Search
- LPDAAC - Getting Started with GEDI L2A v2 Data in Python
- The GEDI Website
- GEDI_L2A v2 User Guide

2.2.4 Importing and Installing Packages

We will start by importing the packages which will allow us to search for, access, explore, and visualize GEDI_L2A product data.

Note: This Jupyter notebook utilizes the `folium` package. If you do not have this installed, uncomment the line and run the following code block.

```
[ ]: # !pip install folium
```

For this tutorial, we will import `boto3`, `folium`, `h5py`, `pandas`, and `exists` from `os.path` as shown in the following codeblock.

[2]:
```
# Install packages
import boto3
import folium
import h5py
import os
import pandas as pd
from maap.maap import MAAP
from os.path import exists
```

2.2.5 Search for GEDI_L2A Data

To search for data from the GEDI_L2A product using NASA's CMR, we invoke the `MAAP` constructor, setting the `maap_host` argument to `'api.maap-project.org'`.

[ ]:
```
# Invoke the MAAP using the MAAP host argument
maap = MAAP(maap_host='api.maap-project.org')
```

Now we can use the `searchGranule` function to find granule data within the collection, using the collection short name "GEDI02_A". Note that we can use `searchGranule`'s `cmr_host` argument to specify a CMR instance external to MAAP and the `readable_granule_name` argument to find granules matching either granule UR or producer granule id (please see the API documentation for more information). In order to download data from NASA's CMR, we will set a variable to the first result from the `results` we obtained.

[4]:
```
# Search for granule data using CMR host name and collection short name, and readable_
↪granule_name arguments
results = maap.searchGranule(
    cmr_host='cmr.earthdata.nasa.gov',
    short_name='GEDI02_A',
    readable_granule_name = "GEDI02_A_2021272190541_O15849_04_T03030_02_003_02_V002.h5")
```

We'll go ahead and create a new data directory (if it doesn't already exist), and download the first result into it.

[12]:
```
# set data directory
dataDir = './data'

# check if directory exists -> if directory doesn't exist, directory is created
if not os.path.exists(dataDir):
    os.mkdir(dataDir)

# Download first result
filename = results[0].getData(dataDir)
```

If desired, the `print` function can be utilized to see the file name and directory.

[13]:
```
# Print file directory
print(filename)
```

```
./data/GEDI02_A_2021272190541_O15849_04_T03030_02_003_02_V002.h5
```

2.2.6 Explore

Now that we have downloaded the data, let's look into what it contains.

[14]:
```
# Create variable containing info from the file we downloaded
gediL2A = h5py.File(filename, 'r')
```

GEDI_L2A data has data for 8 different beams. Let's create a list of beam names to help explore the data.

[15]:
```
# Create list of beam names
beamNames = [g for g in gediL2A.keys() if g.startswith('BEAM')]
beamNames
```

[15]:
```
['BEAM0000',
 'BEAM0001',
 'BEAM0010',
 'BEAM0011',
 'BEAM0101',
 'BEAM0110',
 'BEAM1000',
 'BEAM1011']
```

Now let's explore the information available for one of the beams (in this case 'BEAM0000').

[16]:
```
# Get list of objects in the data pertaining to 'BEAM0000'
beam = beamNames[0]
gediL2A_objs = []
gediL2A.visit(gediL2A_objs.append)
gediSDS = [o for o in gediL2A_objs if isinstance(gediL2A[o], h5py.Dataset)]
[i for i in gediSDS if beam in i][0:20]
```

[16]:
```
['BEAM0000/ancillary/l2a_alg_count',
 'BEAM0000/beam',
 'BEAM0000/channel',
 'BEAM0000/degrade_flag',
 'BEAM0000/delta_time',
 'BEAM0000/digital_elevation_model',
 'BEAM0000/digital_elevation_model_srtm',
 'BEAM0000/elev_highestreturn',
 'BEAM0000/elev_lowestmode',
 'BEAM0000/elevation_bias_flag',
 'BEAM0000/elevation_bin0_error',
 'BEAM0000/energy_total',
 'BEAM0000/geolocation/elev_highestreturn_a1',
 'BEAM0000/geolocation/elev_highestreturn_a2',
 'BEAM0000/geolocation/elev_highestreturn_a3',
 'BEAM0000/geolocation/elev_highestreturn_a4',
 'BEAM0000/geolocation/elev_highestreturn_a5',
 'BEAM0000/geolocation/elev_highestreturn_a6',
 'BEAM0000/geolocation/elev_lowestmode_a1',
 'BEAM0000/geolocation/elev_lowestmode_a2']
```

2.2.7 Visualize

Now that we've seen the various labels within the /BEAM0000 group, let's use this information to visualize the GEDI orbit path for our scenes. To start, we shall get samples for various shots, the beam number, longitude, latitude, and quality flags. We can use these samples to create and display a `pandas` dataframe.

[19]:
```
# Set variables for shot, beam number, longitude, latitude, and quality flag samples
lonSample, latSample, shotSample, qualitySample, beamSample = [], [], [], [], []
lats = gediL2A[f'{beamNames[0]}/lat_lowestmode'][()]
lons = gediL2A[f'{beamNames[0]}/lon_lowestmode'][()]
shots = gediL2A[f'{beamNames[0]}/shot_number'][()]
quality = gediL2A[f'{beamNames[0]}/quality_flag'][()]
for i in range(len(shots)):
    if i % 100 == 0:
        shotSample.append(str(shots[i]))
        lonSample.append(lons[i])
        latSample.append(lats[i])
        qualitySample.append(quality[i])
        beamSample.append(beamNames[0])

# Create a pandas dataframe containing the sample information
latslons = pd.DataFrame({'Beam': beamSample, 'Shot Number': shotSample, 'Longitude':␣
↪lonSample,
                         'Latitude': latSample, 'Quality Flag': qualitySample})

# Display the dataframe
latslons
```

[19]:

	Beam	Shot Number	Longitude	Latitude	Quality Flag
0	BEAM0000	158490000400502383	52.337698	0.599011	0
1	BEAM0000	158490000400502483	52.367554	0.556129	0
2	BEAM0000	158490000400502583	52.396932	0.514800	0
3	BEAM0000	158490000400502683	52.426548	0.472746	0
4	BEAM0000	158490000400502783	52.456248	0.430695	0
...	...	...	...	...	...
1685	BEAM0000	158490000400670883	135.837810	-51.605016	0
1686	BEAM0000	158490000400670983	135.921291	-51.605653	0
1687	BEAM0000	158490000400671083	136.003206	-51.606299	0
1688	BEAM0000	158490000400671183	136.086629	-51.606741	0
1689	BEAM0000	158490000400671283	136.168523	-51.607159	0

[1690 rows x 5 columns]

We can now create a map of the orbit path using the dataframe that we have created and utilizing the `folium.Map` and `folium.Circle` functions. Include `map` in the code block to inspect the map which is created within the Jupyter notebook.

[20]:
```
# Create a map
map = folium.Map(
    location=[
        latslons.Latitude.mean(),
        latslons.Longitude.mean()
    ], zoom_start=3, control_scale=True
)
```

(continues on next page)

(continued from previous page)

```
# Add variables to the map
for index, location_info in latslons.iterrows():
    folium.Circle(
        [location_info["Latitude"], location_info["Longitude"]],
        popup=f"Shot Number: {location_info['Shot Number']}"
    ).add_to(map)

# Display map
map
```

[20]: <folium.folium.Map at 0x7f196e27bb80>

2.3 GEDI_L2B Search and Visualize

Authors: Samuel Ayers (UAH), Sumant Jha (MSFC/USRA), Anish Bhusal (UAH), Alex Mandel (DevSeed), Aimee Barciauskas (DevSeed)

Date: December 19, 2022

Description: In this tutorial, we will use the integrated Earthdata search function in MAAP Algorithm Development Environment (ADE) to search for GEDI L2B data for an area of interest. We will then download some of this data and read its attributes in preparation for some analysis. We will perform a spatial subset on the data to reduce data volumes, and then make some basic plots of our data.

2.3.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.3.2 About the Data

GEDI L2B Canopy Cover and Vertical Profile Metrics Data Global Footprint Level V002

This dataset provides Global Ecosystem Dynamics Investigation (GEDI) Level 2 (L2) data, which has the purpose of extracting biophysical metrics and consists of Canopy Cover and Vertical Profile Metrics. These metrics are derived from the L1B waveform, and include canopy cover, Plant Area Index (PAI), Plant Area Volume Density (PAVD), and Foliage Height Diversity (FHD). GEDI is attached to the International Space Station (ISS) and collects data globally between 51.6° N and 51.6° S latitudes at the highest resolution and densest sampling of any light detection and ranging (lidar) instrument in orbit to date; specifically, GEDI L2B data has a spatial resolution of 25m. (Source: GEDI L2B CMR Search)

2.3.3 Additional Resources

- GEDI_L2B Version 2 Dataset Landing Page
- GEDI Level 2 Version 2 User Guide
- The GEDI Website

2.3.4 Importing and Installing Packages

We will begin by installing any packages we need and importing the packages that we will use.

Prerequisites

- geopandas
- folium

[ ]:
```
# Uncomment the following lines to install these packages if you haven't already.
# !pip install geopandas
# !pip install folium
```

[38]:
```
from maap.maap import MAAP
maap = MAAP(maap_host='api.maap-project.org')
import geopandas as gpd
import folium
import h5py
import pandas
import matplotlib
import matplotlib.pyplot as plt
import shapely
import os
```

2.3.5 Search Data Using an AOI

We will search and download GEDI L2B data using the bounding box of a vector AOI. Firstly, an AOI over the Shenandoah National Park will be created and then we will plot its location on a map.

[39]:
```
# Using bounding coordinates to create a polygon around Shenandoah National Park
lon_coords = [-78.32129105072025, -78.04618813890727, -78.72985973163064, -79.
↪0158578082679, -78.32129105072025]
lat_coords = [38.88703610703791, 38.74909216350823, 37.88789051477522, 38.03177640342157,
↪ 38.88703610703791]

polygon_geom = shapely.geometry.polygon.Polygon(zip(lon_coords, lat_coords))
crs = 'epsg:4326'
AOI = gpd.GeoDataFrame(index=[0], crs=crs, geometry=[polygon_geom])
```

We can get the bounding box of the AOI so we can use it as a spatial limit on our data search. GeoPandas has a function for returning the spatial coordinates of a bounding box:

[40]:
```
# Get the bounding box of the shp
bbox = AOI.bounds
# print the bounding box coords
```

(continues on next page)

(continued from previous page)

```
print('minx = ', bbox['minx'][0])
print('miny = ', bbox['miny'][0])
print('maxx = ', bbox['maxx'][0])
print('maxy = ', bbox['maxy'][0])
```

```
minx =  -79.0158578082679
miny =  37.88789051477522
maxx =  -78.04618813890727
maxy =  38.88703610703791
```

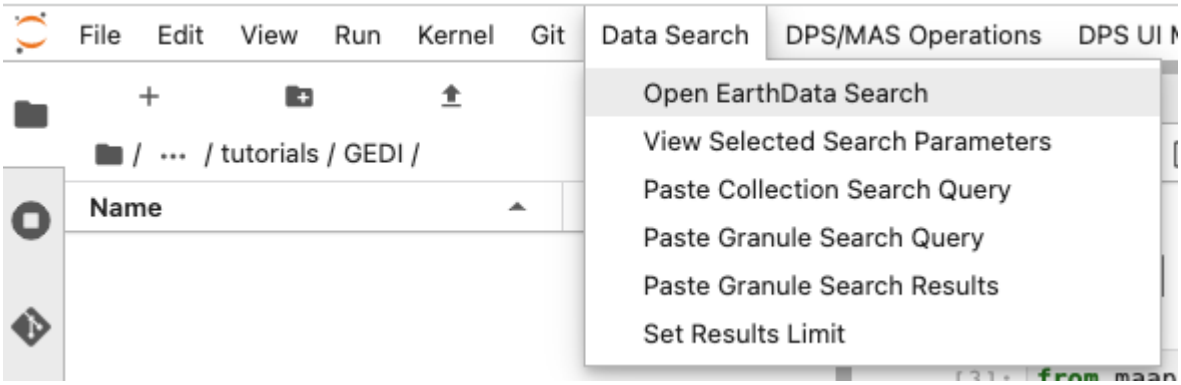
Let's look at our AOI on an interactive map using folium.

[41]:
```
m = folium.Map(location=[38.5, -78], zoom_start=9, tiles='CartoDB positron')
geo_j = folium.GeoJson(data=AOI, style_function=lambda x: {'fillColor': 'orange'})
geo_j.add_to(m)
m
```

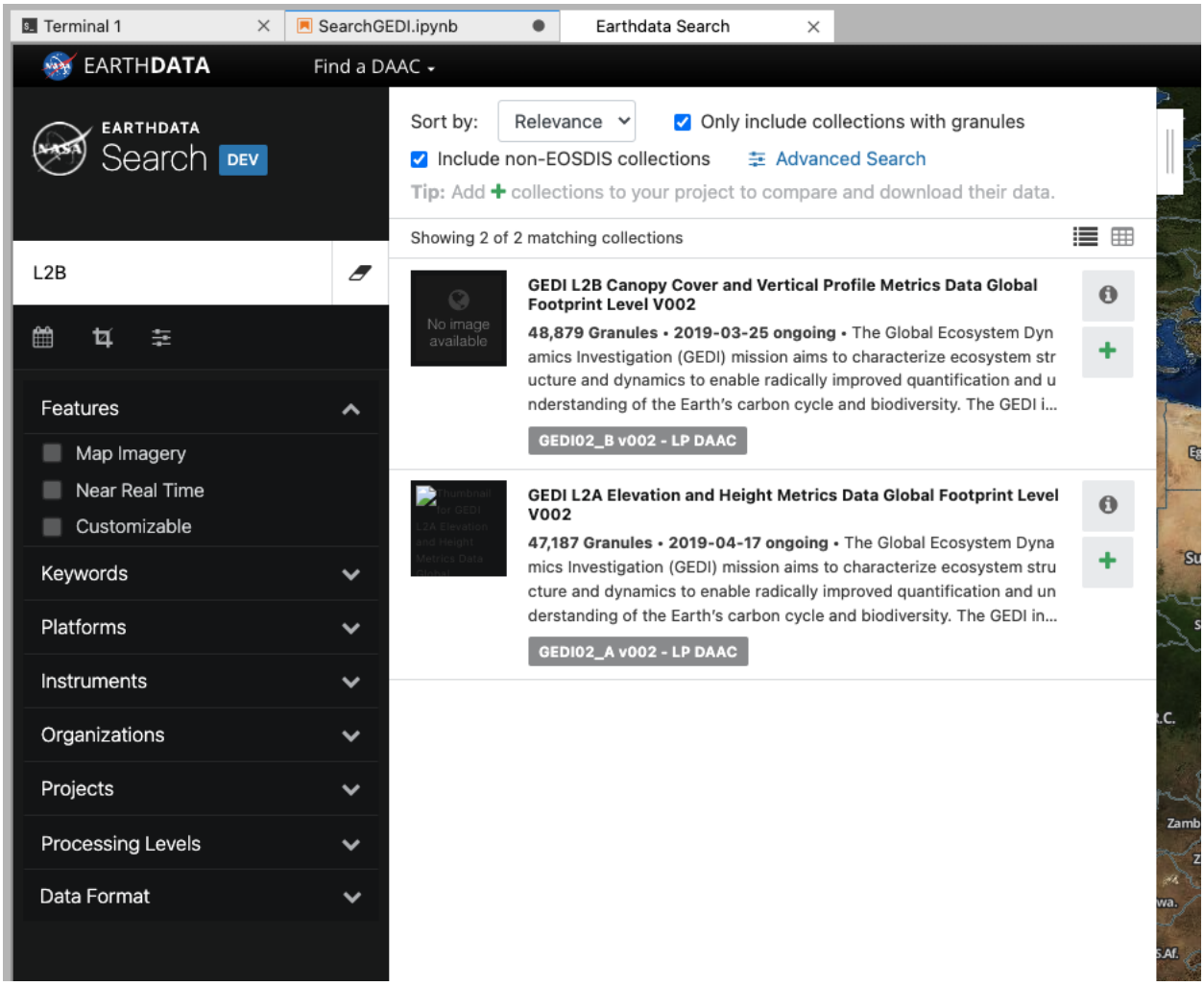
[41]:
```
<folium.folium.Map at 0x7fa48a66b710>
```

2.3.6 Search using the EarthData Search Integration

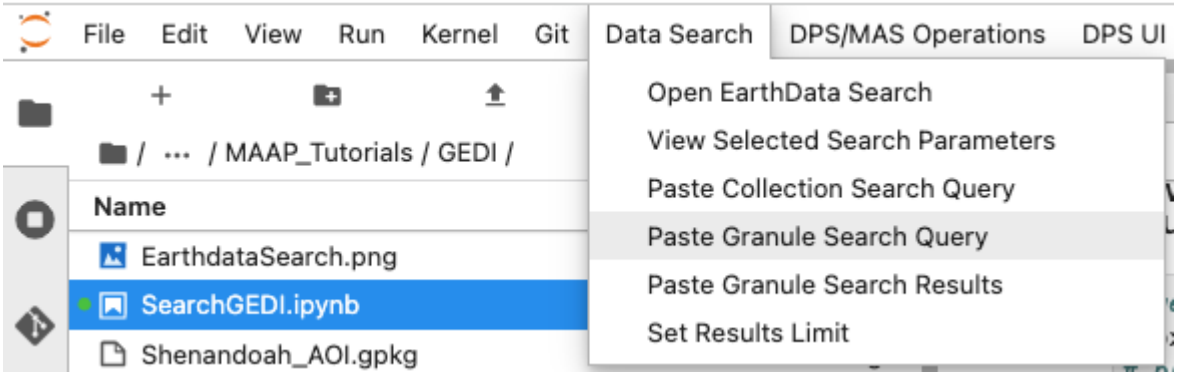
Search Data: To search GEDI data we can use the EarthData search integration in the MAAP ADE. Open the Earthdata search toolbar by clicking on the following:



This will open up the EarthData search interface in a new tab. We can use the search bar to search GEDI L2B data. By entering "L2B" in the search bar, we can see the relevant GEDI data is filtered. Click on the dataset to get more details.



While we have been searching for data, the MAAP ADE has been keeping a track of our search parameters. This means that we can easily insert our search for GEDI data straight into our notebook.



2.3.7 Inspect and Filter through the Data

This gives us our search parameters in our notebook using the GEDI dataset ID. The default limit for the number of returned results is 1000. Running this will produce 1000 results, but we can look at the first one by indexing the list of returned results. This is what the data entry looks like. You can see a large amount of metadata for the file along with the URL for where this specific file is stored.

[42]:
```
data = maap.searchGranule(cmr_host='cmr.earthdata.nasa.gov', concept_id="C1908350066-
↪LPDAAC_ECS", limit=1000)[0]
data
```

[42]:
```
{'concept-id': 'G2011069580-LPDAAC_ECS',
 'collection-concept-id': 'C1908350066-LPDAAC_ECS',
 'revision-id': '10',
 'format': 'application/echo10+xml',
 'Granule': {'GranuleUR': 'SC:GEDI02_B.002:2433465280',
  'InsertTime': '2021-02-22T19:16:30.675Z',
  'LastUpdate': '2021-09-16T13:36:07.965Z',
  'Collection': {'DataSetId': 'GEDI L2B Canopy Cover and Vertical Profile Metrics Data␣
↪Global Footprint Level V002'},
  'DataGranule': {'SizeMBDataGranule': '16.5413',
   'ProducerGranuleId': 'GEDI02_B_2019108002012_O01959_01_T03909_02_003_01_V002.h5',
   'DayNightFlag': 'UNSPECIFIED',
   'ProductionDateTime': '2021-02-21T14:45:08Z'},
  'PGEVersionClass': {'PGEVersion': '003'},
  'Temporal': {'RangeDateTime': {'BeginningDateTime': '2019-04-18T00:20:12.000000Z',
    'EndingDateTime': '2019-04-18T01:52:53.000000Z'}},
  'Spatial': {'HorizontalSpatialDomain': {'Geometry': {'GPolygon': {'Boundary': {'Point':
↪ [{'PointLongitude': '80.2890335089',
        'PointLatitude': '-4.6168623465'},
       {'PointLongitude': '82.4542052313', 'PointLatitude': '-1.5568021223'},
       {'PointLongitude': '83.6402279084', 'PointLatitude': '0.1277767863'},
       {'PointLongitude': '83.6814118126', 'PointLatitude': '0.0987901632'},
       {'PointLongitude': '82.4953794073', 'PointLatitude': '-1.5858244175'},
       {'PointLongitude': '80.3302671214',
        'PointLatitude': '-4.6459691487'}]}}}},
  'OrbitCalculatedSpatialDomains': {'OrbitCalculatedSpatialDomain': {'StartOrbitNumber':
↪'1959',
    'StopOrbitNumber': '1959'}},
  'MeasuredParameters': {'MeasuredParameter': {'ParameterName': 'Level2B'}},
  'Platforms': {'Platform': {'ShortName': 'ISS',
    'Instruments': {'Instrument': {'ShortName': 'GEDI',
      'Sensors': {'Sensor': {'ShortName': 'LIDAR'}}}}}},
  'Campaigns': {'Campaign': {'ShortName': 'GEDI'}},
  'AdditionalAttributes': {'AdditionalAttribute': [{'Name': 'identifier_product_doi',
     'Values': {'Value': '10.5067/GEDI/GEDI02_B.002'}},
    {'Name': 'identifier_product_doi_authority',
     'Values': {'Value': 'https://doi.org'}},
    {'Name': 'Reference_Ground_Track', 'Values': {'Value': '3909'}},
    {'Name': 'SEGMENT_NUMBER', 'Values': {'Value': '01'}}]},
  'InputGranules': {'InputGranule': ['GEDI01_B_2019108002012_O01959_01_T03909_02_005_01_
↪V002.h5',
    'GEDI02_A_2019108002012_O01959_01_T03909_02_003_01_V002_10algs.h5']},
  'OnlineAccessURLs': {'OnlineAccessURL': {'URL': 'https://e4ftl01.cr.usgs.gov//GEDI_L1_
```

(continues on next page)

(continued from previous page)

```
↪L2/GEDI/GEDI02_B.002/2019.04.18/GEDI02_B_2019108002012_O01959_01_T03909_02_003_01_V002.
↪h5',
    'URLDescription': 'GEDI02_B_2019108002012_O01959_01_T03909_02_003_01_V002.h5.␣
↪MimeType: application/x-hdfeos',
    'MimeType': 'application/x-hdfeos'}},
  'OnlineResources': {'OnlineResource': [{'URL': 'https://doi.org/10.5067/GEDI/GEDI02_B.
↪001 ',
     'Description': 'The Landing Page for this file may be accessed directly from this␣
↪link',
     'Type': 'DOI',
     'MimeType': 'text/html'},
    {'URL': 'https://e4ftl01.cr.usgs.gov//WORKING/BRWS/Browse.001/2021.02.23/GEDI02_B_
↪2019108002012_O01959_01_T03909_02_003_01_V002.png',
     'Description': 'This Browse file may be downloaded directly from this link',
     'Type': 'BROWSE',
     'MimeType': 'image/jpeg'},
    {'URL': 'https://e4ftl01.cr.usgs.gov//GEDI_L1_L2/GEDI/GEDI02_B.002/2019.04.18/GEDI02_
↪B_2019108002012_O01959_01_T03909_02_003_01_V002.h5.xml',
     'Description': 'This Metadata file may be downloaded directly from this link',
     'Type': 'EXTENDED METADATA',
     'MimeType': 'text/xml'}]},
  'Orderable': 'true',
  'DataFormat': 'HDF5',
  'Visible': 'true'}}
```

So far, this search function requests all of the GEDI data but we can add a spatial subset filter using our AOI from above to limit the results. Adding a spatial filter returns 259 GEDI L2B files that intersect with our AOI.

[43]:
```
data_aoi = maap.searchGranule(cmr_host='cmr.earthdata.nasa.gov', concept_id="C1908350066-
↪LPDAAC_ECS", bounding_box="-79.0158578082679,37.88789051477522,-78.04618813890727,38.
↪887036107037915", limit=1000)
print(len(data_aoi))
```

```
259
```

This is more data than we need, so let's look at the contents of a single GEDI file. Firstly we need to bring it from the server side (S3) to our local side. This can be done using the MAAP function getData. We'll create a new data directory, and then we will pull the 7th file in our search results.

[44]:
```
# set data directory
dataDir = './data'

# check if directory exists -> if directory doesn't exist, directory is created
if not os.path.exists(dataDir):
    os.mkdir(dataDir)
```

[45]:
```
# pulling the 7th file into the new directory
gedi_data = data_aoi[6].getData(dataDir)
print(gedi_data)
```

```
./data/GEDI02_B_2019145051352_O02537_03_T04809_02_003_01_V002.h5
```

GEDI data has 8 beams. So, we will check that all beams are in our file and print a list of the available beams.

[46]:
```
gedi_h5_file = h5py.File(gedi_data, 'r')
gedi_keys = list(gedi_h5_file.keys())
gedi_beams = ['BEAM0000', 'BEAM0001', 'BEAM0010', 'BEAM0011', 'BEAM0101', 'BEAM0110',
↪'BEAM1000', 'BEAM1011']
gedi_beams_lst = []
for gedi_beam_name in gedi_keys:
    if gedi_beam_name in gedi_beams:
        gedi_beams_lst.append(gedi_beam_name)
print(gedi_beams_lst)
```

```
['BEAM0000', 'BEAM0001', 'BEAM0010', 'BEAM0011', 'BEAM0101', 'BEAM0110', 'BEAM1000',
↪'BEAM1011']
```

For each beam, we need to get all of the data and metrics associated with it. For this, we have a function that will gather all of the metrics we want and put them into a geopandas dataframe:

[47]:
```
def get_gedi_df(gedi_h5_file, gedi_beam_name):
    gedi_beam = gedi_h5_file[gedi_beam_name]

    # Get location info.
    gedi_beam_geoloc = gedi_beam['geolocation']
    # Get land cover data.
    gedi_beam_landcover = gedi_beam['land_cover_data']

    gedi_beam_df = pandas.DataFrame(
            {'elevation_bin0'         : gedi_beam_geoloc['elevation_bin0'],
             'elevation_lastbin'      : gedi_beam_geoloc['elevation_lastbin'],
             'height_bin0'            : gedi_beam_geoloc['height_bin0'],
             'height_lastbin'         : gedi_beam_geoloc['height_lastbin'],
             'shot_number'            : gedi_beam_geoloc['shot_number'],
             'solar_azimuth'          : gedi_beam_geoloc['solar_azimuth'],
             'solar_elevation'        : gedi_beam_geoloc['solar_elevation'],
             'latitude_bin0'          : gedi_beam_geoloc['latitude_bin0'],
             'latitude_lastbin'       : gedi_beam_geoloc['latitude_lastbin'],
             'longitude_bin0'         : gedi_beam_geoloc['longitude_bin0'],
             'longitude_lastbin'      : gedi_beam_geoloc['longitude_lastbin'],
             'degrade_flag'           : gedi_beam_geoloc['degrade_flag'],
             'digital_elevation_model': gedi_beam_geoloc['digital_elevation_model'],
             'landsat_treecover'      : gedi_beam_landcover['landsat_treecover'],
             'modis_nonvegetated'     : gedi_beam_landcover['modis_nonvegetated'],
             'modis_nonvegetated_sd'  : gedi_beam_landcover['modis_nonvegetated_sd'],
             'modis_treecover'        : gedi_beam_landcover['modis_treecover'],
             'modis_treecover_sd'     : gedi_beam_landcover['modis_treecover_sd'],
             'beam'                   : gedi_beam['beam'],
             'cover'                  : gedi_beam['cover'],
             'master_frac'            : gedi_beam['master_frac'],
             'master_int'             : gedi_beam['master_int'],
             'num_detectedmodes'      : gedi_beam['num_detectedmodes'],
             'omega'                  : gedi_beam['omega'],
             'pai'                    : gedi_beam['pai'],
             'pgap_theta'             : gedi_beam['pgap_theta'],
             'pgap_theta_error'       : gedi_beam['pgap_theta_error'],
             'rg'                     : gedi_beam['rg'],
```

(continues on next page)

(continued from previous page)

```
            'rh100'                 : gedi_beam['rh100'],
            'rhog'                  : gedi_beam['rhog'],
            'rhog_error'            : gedi_beam['rhog_error'],
            'rhov'                  : gedi_beam['rhov'],
            'rhov_error'            : gedi_beam['rhov_error'],
            'rossg'                 : gedi_beam['rossg'],
            'rv'                    : gedi_beam['rv'],
            'sensitivity'           : gedi_beam['sensitivity'],
            'stale_return_flag'     : gedi_beam['stale_return_flag'],
            'surface_flag'          : gedi_beam['surface_flag'],
            'l2a_quality_flag'      : gedi_beam['l2a_quality_flag'],
            'l2b_quality_flag'      : gedi_beam['l2b_quality_flag']})

    gedi_beam_gdf = gpd.GeoDataFrame(gedi_beam_df, crs='EPSG:4326',
                                     geometry=gpd.points_from_xy(gedi_beam_df.
↪longitude_lastbin,
                                                                 gedi_beam_
↪df.latitude_lastbin))
    return gedi_beam_gdf
```

To access the data with this function, we can call the function for each beam id that we have:

[48]:
```
BEAM0000 = get_gedi_df(gedi_h5_file, 'BEAM0000')
BEAM0001 = get_gedi_df(gedi_h5_file, 'BEAM0001')
BEAM0010 = get_gedi_df(gedi_h5_file, 'BEAM0010')
BEAM0011 = get_gedi_df(gedi_h5_file, 'BEAM0011')
BEAM0101 = get_gedi_df(gedi_h5_file, 'BEAM0101')
BEAM0110 = get_gedi_df(gedi_h5_file, 'BEAM0110')
BEAM1000 = get_gedi_df(gedi_h5_file, 'BEAM1000')
BEAM1011 = get_gedi_df(gedi_h5_file, 'BEAM1011')
```

Now we can look at the data in one of the dataframes (beams). We can see that there are 108583 GEDI shots (108583 entries in each column).

[49]:
```
print(BEAM0000.head())
print('number of rows = ', len(BEAM0000))
```

```
   elevation_bin0  elevation_lastbin  height_bin0  height_lastbin  \
0      533.320205         532.870899 -9999.000000    -9999.000000
1      533.610735         533.161429 -9999.000000    -9999.000000
2      835.586133         790.954687    42.284134       -2.373195
3      822.361914         780.875651    38.280968       -3.229356
4      788.459662         760.602475    25.957445       -1.915897

         shot_number  solar_azimuth  solar_elevation  latitude_bin0  \
0  25370000300165164     -34.511044       -10.500773      51.821288
1  25370000300165165     -34.510292       -10.501078      51.821282
2  25370000300165166     -34.509689       -10.501341      51.821258
3  25370000300165167     -34.508926       -10.501648      51.821253
4  25370000300165168     -34.508163       -10.501955      51.821249

   latitude_lastbin  longitude_bin0  ...  rhov  rhov_error  rossg  \
0         51.821288     -127.096372  ...   0.6     -9999.0    0.5
```

(continues on next page)

(continued from previous page)

```
1         51.821282     -127.095550  ...   0.6     -9999.0    0.5
2         51.821260     -127.094876  ...   0.6     -9999.0    0.5
3         51.821255     -127.094048  ...   0.6     -9999.0    0.5
4         51.821250     -127.093210  ...   0.6     -9999.0    0.5

            rv  sensitivity  stale_return_flag  surface_flag  \
0 -9999.000000   -14.502214                  1             1
1 -9999.000000    -8.689309                  1             1
2  3615.732178     0.965246                  0             1
3  4290.229980     0.969620                  0             1
4  2557.771240     0.943713                  0             1

   l2a_quality_flag  l2b_quality_flag                     geometry
0                 0                 0  POINT (-127.09637 51.82129)
1                 0                 0  POINT (-127.09555 51.82128)
2                 1                 1  POINT (-127.09485 51.82126)
3                 1                 1  POINT (-127.09403 51.82126)
4                 1                 1  POINT (-127.09320 51.82125)

[5 rows x 41 columns]
number of rows =  108583
```

2.3.8 Create the Subset and Display the Data

Before displaying this data we can do a spatial subset to remove the data outside of our AOI by doing a spatial subset. We use the Geopandas clip function to clip out the GEDI data based on the extent of our AOI. This reduces the size of the dataframe from 108583 rows to 508.

[50]:
```
BEAM0000_sub = gpd.clip(BEAM0000, AOI)
len(BEAM0000_sub)
```

[50]:
```
508
```

We can display this subset of data over our AOI extent.

[51]:
```
m = folium.Map(location=[38.5, -78], zoom_start=9, tiles='CartoDB positron')

geo_j = folium.GeoJson(data=AOI, style_function=lambda x: {'fillColor': 'orange'})
geo_j.add_to(m)

geo_g = folium.GeoJson(data=BEAM0000_sub,marker = folium.CircleMarker(radius = 1, #␣
↪Radius in metres
                                                      weight = 0, #outline weight
                                                      fill_color = '#FF0000',
                                                      fill_opacity = 1))
geo_g.add_to(m)
m
```

[51]:
```
<folium.folium.Map at 0x7fa48b58f890>
```

Now we have this subset of data, we can look at some of the GEDI metrics over our area of interest. The 'rh100' metric should give us the top of the canopy heights. What does this look like over Shenandoah National Park? We will look at the ground height and the 'rh100' above ground. The DEM metric is in meters (m) and the 'rh100' metric is in

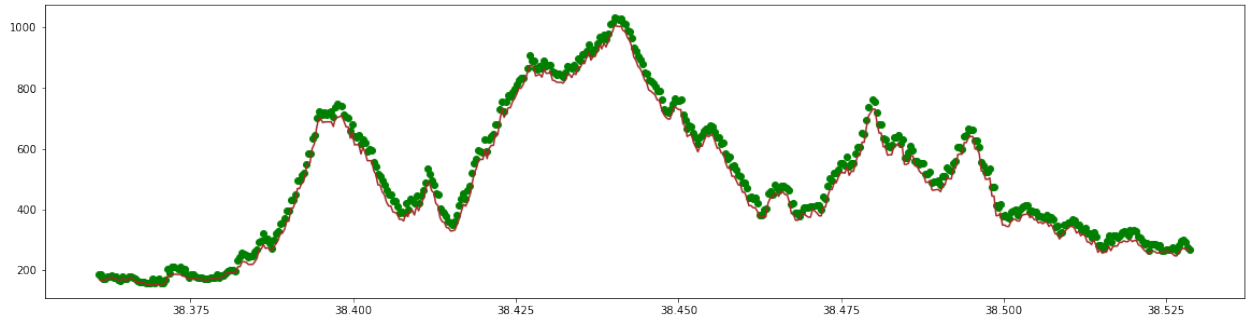
centimeters (cm) above the ground height. Therefore we must add the 'rh100' value to the ground height to get the total height of the tree for display purposes. The 'rh100' metric is also converted to meters (m) to normalize the units

[52]:
```
latitude = BEAM0000_sub['latitude_lastbin']
rh100 = BEAM0000_sub['rh100']
ground = BEAM0000_sub['digital_elevation_model']

TreeHeight = ground + rh100/100
```

Finally, we can make a plot of the ground surface and the canopy heights above the ground surface. We can see the forest canopies in green above the topographically complex ground in brown.

[53]:
```
plt.figure(figsize=(20, 5))
plt.scatter(latitude, TreeHeight, c='green')
plt.plot(latitude, ground, c='brown')
```

[53]:
```
[<matplotlib.lines.Line2D at 0x7fa48b7eb7d0>]
```



Now you have this basic structure you can investigate some of the other metrics and GEDI beams to understand more about the data.

2.4 GEDI_L3 Search and Download

Authors: Anish Bhusal (UAH), Sumant Jha (MSFC/USRA), Jamison French (DevSeed), Aimee Barciauskas (DevSeed), Alex Mandel (DevSeed)

Date: February 8, 2023

Description: In this tutorial, we will search for GEDI L3 data in NASA's Earthdata Search, and then download a GeoTIFF file from the ORNL DAAC S3. We will then visualize the GeoTIFF file, which consists of canopy heights.

2.4.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

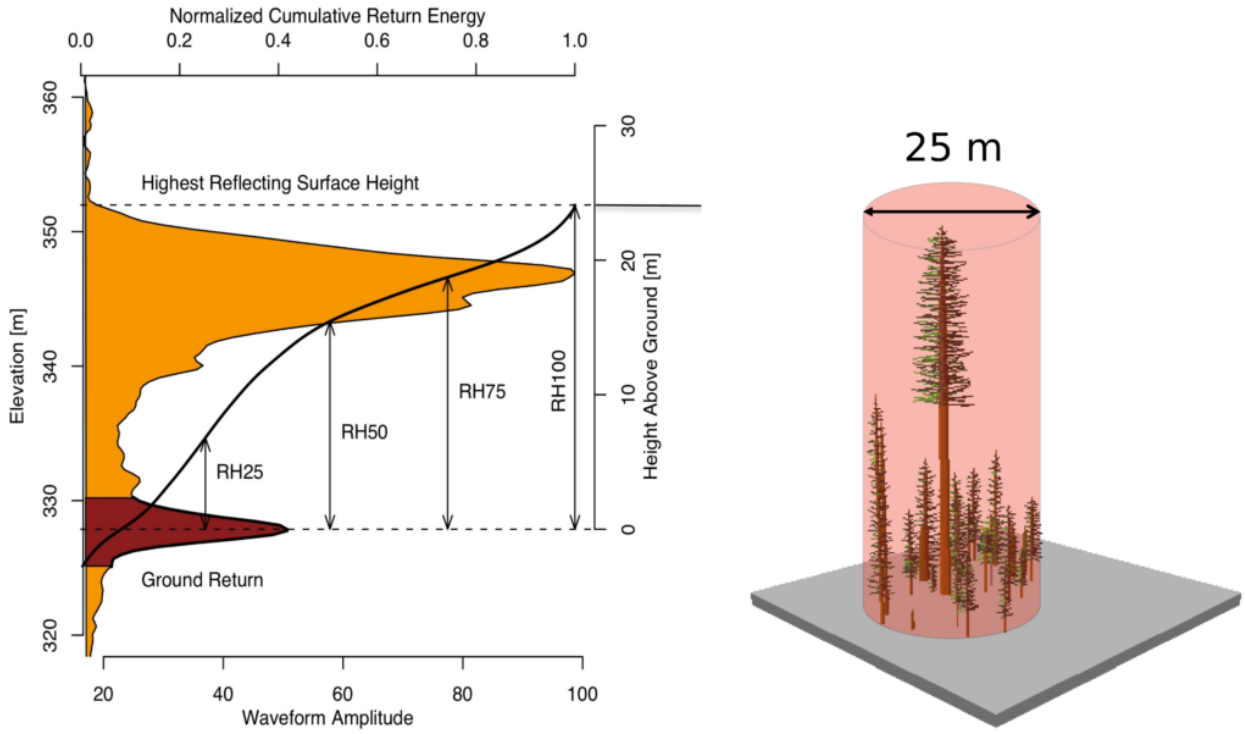
2.4.2 About the Data

GEDI L3 Gridded Land Surface Metrics, Version 2

This dataset provides Global Ecosystem Dynamics Investigation (GEDI) Level 3 (L3) gridded mean canopy height, standard deviation of canopy height, mean ground elevation, standard deviation of ground elevation, and counts of laser footprints per 1-km x 1-km grid cells globally within -52 and 52 degrees latitude. GEDI is attached to the International Space Station (ISS) and collects data globally between 51.6° N and 51.6° S latitudes at the highest resolution and densest sampling of any light detection and ranging (lidar) instrument in orbit to date.

GEDI L3 data products are gridded by spatially interpolating Level 2 footprint estimates of canopy cover, canopy height, Leaf Area Index (LAI), vertical foliage profile and their uncertainties. Level 2 data contains terrain elevation, canopy height, RH metrics and Leaf Area Index (LAI). The raw waveforms are collected by GEDI system and processed to provide canopy height and profile metrics. These metrics provide easy to use and interpret information about the vertical distribution of canopy material.

Source: GEDI L3 Gridded Land Surface Metrics, Version 2 Data Set Landing Page



Source: https://gedi.umd.edu/data/products/

Figure: Sample of GEDI lidar waveform (left). The light brown area under the curve represents the return energy from the canopy, while the dark brown area signifies the return from the underlying typography. The black line is cumulative return energey, starting from the bottom of the ground return (normalized to 0) to the top of canopy (normalized to 1). The diagram on the right shows the distribution of trees that produced the waveform.

2.4.3 Additional Resources

- GEDI L3 Gridded Land Surface Metrics, Version 2 User Guide
- GEDI Overview Page, LPDAAC

2.4.4 Importing and Installing Packages

This tutorial assumes you've all packages installed. If you haven't already, uncomment the following lines to install these packages.

[ ]:
```
# !pip install geopandas
# !pip install contextily
# !pip install backoff
# !pip install folium
# !pip install geojsoncontour
```

[16]:
```
from maap.maap import MAAP
import pandas as pd
import folium
from rasterio.plot import show
import rasterio
import boto3
import os
```

After importing necessary packages, the next step is to initialize MAAP constructor using `api.maap-project.org` as `maap_host` argument.

[17]:
```
maap = MAAP(maap_host="api.maap-project.org")
```

Now, the next step is to seach granules from the CMR. To generate following query, you can use EarthData search feature from MAAP ADE. Refer to this tutorial for more info.

[18]:
```
results=maap.searchGranule(cmr_host='cmr.earthdata.nasa.gov',concept_id="C2153683336-
↪ORNL_CLOUD", limit=1000)
```

The above query gives 1000 results by default. The number of necessary results can be changed using `limit` argument. We can view the `GranuleUR` from results using:

[19]:
```
[result['Granule']['GranuleUR'] for result in results]
```

[19]:
```
['GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_stddev_2019108_2020287_002_02.tif
↪',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_counts_2019108_2020287_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_counts_2019108_2021104_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_mean_2019108_2020287_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_stddev_2019108_2020287_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_stddev_2019108_2021104_002_02.tif
↪',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_stddev_2019108_2021104_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_mean_2019108_2020287_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_mean_2019108_2021104_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_mean_2019108_2021104_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_stddev_2019108_2021216_002_02.tif
```

(continues on next page)

(continued from previous page)

```
↪',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_mean_2019108_2021216_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_stddev_2019108_2021216_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_mean_2019108_2021216_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_counts_2019108_2021216_002_02.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_mean_2019108_2022019_002_03.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_counts_2019108_2022019_002_03.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_mean_2019108_2022019_002_03.tif',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_elev_lowestmode_stddev_2019108_2022019_002_03.tif
↪',
 'GEDI_L3_LandSurface_Metrics_V2.GEDI03_rh100_stddev_2019108_2022019_002_03.tif']
```

Before downloading a particular tif, let's catch the collection and file name for first item in results:

[20]:
```
granule_ur=results[0]['Granule']['GranuleUR'].split(".")
collection_name=granule_ur[0]
file_name=granule_ur[1]
```

[21]:
```
print(f"collection name: {collection_name} | file_name: {file_name}")
```

```
collection name: GEDI_L3_LandSurface_Metrics_V2 | file_name: GEDI03_elev_lowestmode_
↪stddev_2019108_2020287_002_02
```

2.4.5 Download file from ORNL DAAC S3

To download the file from the source, temporary s3 credentials are required for `maap` package. You can explicitly request `s3_cred_endpoint` for the credentials. The code below wraps that request to get credentials and download the file to your workspace.

[22]:
```
def get_s3_creds(url):
    return maap.aws.earthdata_s3_credentials(url)

def get_s3_client(s3_cred_endpoint):
    creds=get_s3_creds(s3_cred_endpoint)
    boto3_session = boto3.Session(
            aws_access_key_id=creds['accessKeyId'],
            aws_secret_access_key=creds['secretAccessKey'],
            aws_session_token=creds['sessionToken']
    )
    return boto3_session.client("s3")

def download_s3_file(s3, bucket, collection_name, file_name):
    os.makedirs("/projects/gedi_l3", exist_ok=True) # create directories, as necessary
    download_path=f"/projects/gedi_l3/{file_name}.tif"
    s3.download_file(bucket, f"gedi/{collection_name}/data/{file_name}.tif", download_
↪path)
    return download_path
```

[23]:
```
s3_cred_endpoint= 'https://data.ornldaac.earthdata.nasa.gov/s3credentials'
s3=get_s3_client(s3_cred_endpoint)
```

[24]:
```
bucket="ornl-cumulus-prod-protected"
download_path=download_s3_file(s3, bucket, collection_name, file_name)
download_path
```

[24]:
```
'/projects/gedi_l3/GEDI03_elev_lowestmode_stddev_2019108_2020287_002_02.tif'
```

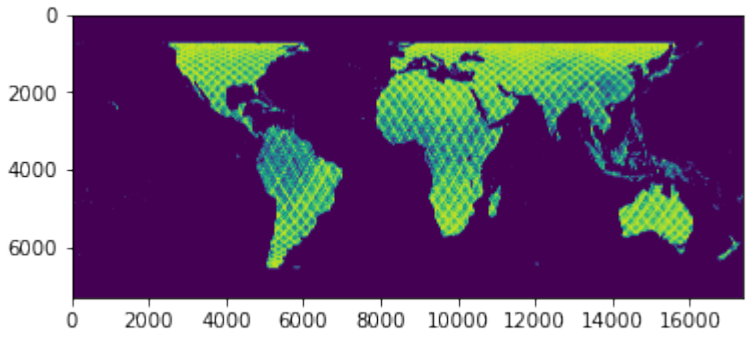
Now, we have the file in our local workspace. It's time to visualize it using `rasterio` package

2.4.6 [Optional] Visualization using Rasterio

The downloaded file is too big to read and visualize directly so we might need to scale it down and view it as a small thumbnail.

[25]:
```
def show_thumbnail(path):
    src=rasterio.open(path)
    oview = src.overviews(1)[0]
    thumbnail = src.read(1, out_shape=(1, int(src.height // oview), int(src.width //␣
↪oview)))
    show(thumbnail)
```

[26]:
```
show_thumbnail(download_path)
```



2.4.7 [Optional] Overlay Raster Layer on top of Folium Map

To properly visualize the canopy heights, we need to display the TIF image on the map. The TIF image file may be too memory and compute-intensive for the kernel causing the process to exit.

[27]:
```
# tif=rasterio.open(download_path)
# arr=tif.read()
# bounds=tif.bounds
```

[28]:
```
#import numpy as np

# x1,y1,x2,y2=bounds
# bbox=[(bounds.bottom, bounds.left), (bounds.top, bounds.right)]
# m=folium.Map(location=[14.59, 120.98], zoom_start=10)
# img = folium.raster_layers.ImageOverlay(image=np.moveaxis(arr, 0, -1), bounds=bbox,␣
↪opacity=0.9, interactive=True, cross_origin=False, zindex=1)
# m
```

2.5 GEDI_L4A Subset and Visualize

Authors: Chuck Daniels (DevSeed), Jamison French (DevSeed), Anish Bhusal (UAH), Sumant Jha (MSFC/USRA), Alex Mandel (DevSeed)

Date: November 14, 2022

Description: In this tutorial, we will use a GeoJSON to create an area of interest (AOI) and use it in MAAP's GEDI Subsetter. We will then visualize the output file created by the subsetter by plotting elevation contours.

2.5.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.5.2 About the Data

GEDI L4A Footprint Level Aboveground Biomass Density, Version 2

This dataset provides Global Ecosystem Dynamics Investigation (GEDI) Level 4 (L4) data, which has the purpose of providing mean aboveground biomass density (AGBD) and consists of the GEDI_L4A and GEDI_L4B collections. GEDI L4A contains predictions of AGBD and estimates of the prediction standard error. GEDI is attached to the International Space Station (ISS) and collects data globally between 51.6° N and 51.6° S latitudes at the highest resolution and densest sampling of any light detection and ranging (lidar) instrument in orbit to date; specifically, GEDI L4A data has a spatial resolution of 25m. Source: GEDI_L4A Version 2 User Guide

2.5.3 Additional Resources

- GEDI_L4A Version 2 Data Set Landing Page
- The GEDI Website
- Earthdata Search

2.5.4 [Optional] Install Python Packages

This notebook contains some cells marked as optional, meaning that you can use this notebook without necessarily running such cells.

However, if you do wish to run the optional cells, you must install the following Python packages, which might not already be installed in your environment:

- `geopandas`: for reading your AOI (GeoJson file), as well as for reading the job output (GeoPackage file containing the subset)
- `contextily`: for visually verifying your AOI
- `backoff`: for repeatedly polling the job status (after submission) until the job has been completed (either successfully or not)
- `folium`: for visualizing your data on a Leaflet map
- `geojsoncontour`: for converting your matplotlib contour plots to geojson

[ ]:
```
# Uncomment the following lines to install these packages if you haven't already.
# !pip install geopandas
# !pip install contextily
# !pip install backoff
# !pip install folium
# !pip install geojsoncontour
```

A job can be submitted without these packages, but installing them in order to run the optional cells may make it more convenient for you to visually verify both your AOI and the subset output produced by your job.

2.5.5 Obtain Username

[18]:
```
from maap.maap import MAAP

maap = MAAP(maap_host="api.maap-project.org")
username = maap.profile.account_info()["username"]
username
```

```
WARNING:maap.maap:Unable to load config file from source maap.cfg
WARNING:maap.maap:Unable to load config file from source ./maap.cfg
WARNING:maap.maap:Unable to load config file from source /projects/maap.cfg
```

[18]:
```
'smk0033'
```

2.5.6 Define the Area of Interest

You may use either a publicly available GeoJSON file for your AOI, such as those available at geoBoundaries, or you may create a custom GeoJSON file for your AOI. The following 2 subsections cover both cases.

Using a geoBoundary GeoJSON File

If your AOI is a publicly available geoBoundary, you can obtain the URL for the GeoJSON file using the function below. You simply need to supply an ISO3 value and a level. To find the appropriate ISO3 and level values, see the table on the geoBoundaries site.

[19]:
```
import requests


def get_geo_boundary_url(iso3: str, level: int) -> str:
    response = requests.get(
        f"https://www.geoboundaries.org/api/current/gbOpen/{iso3}/ADM{level}"
    )
    response.raise_for_status()
    return response.json()["gjDownloadURL"]


# If using a geoBoundary, uncomment the following assignment, supply
# appropriate values for `<iso3>` and `<level>`, then run this cell.

# Example (Gabon level 0): get_geo_boundary("GAB", 0)
```

(continues on next page)

(continued from previous page)

```
# aoi = get_geo_boundary_url("<iso3>", <level>)
```

Using a Custom GeoJSON File

Alternatively, you can make your own GeoJSON file for your AOI and place it within your `my-public-bucket` folder within the ADE.

Based upon where you place your GeoJSON file under `my-public-bucket`, you can construct the URL for a job's `aoi` input value.

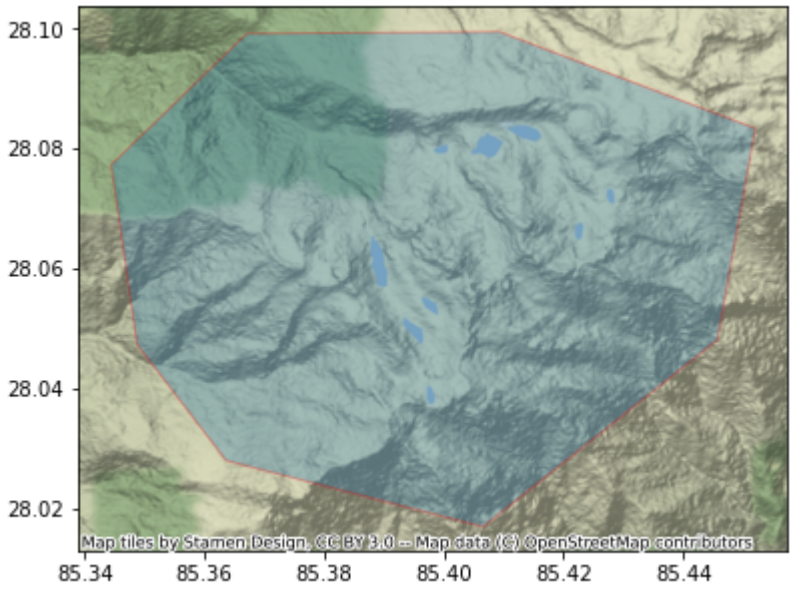
For example, if the relative path of your AOI GeoJSON file under `my-public-bucket` is `path/to/my-aoi.geojson` (avoid using whitespace in the path and filename), the URL you would supply as the value of a job's `aoi` input would be the following (where `{username}` is replaced with your username as output from the previous section):

```
f"https://maap-ops-workspace.s3.amazonaws.com/shared/{username}/path/to/my-aoi.geojson"`
```

If this is the case, use the cell below.

[20]:
```
#aoi = f"https://maap-ops-workspace.s3.amazonaws.com/shared/{username}/langtang_np.
↪geojson"

#for your convenience you can use this geoJSON file but if you have your own geojson,␣
↪use the commented link as example format
aoi = f"https://maap-ops-workspace.s3.amazonaws.com/shared/anisbhsl/langtang_np.geojson"
```

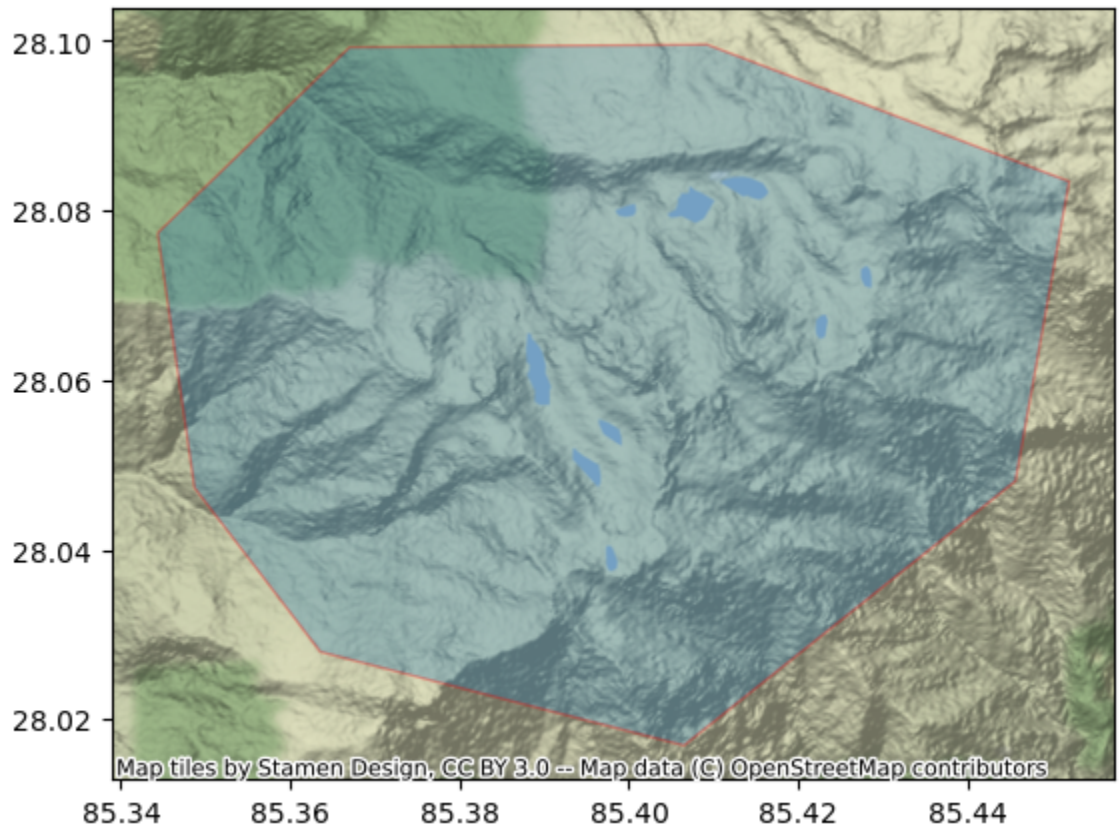
This example uses the AOI of Gosaikunda Lake region inside Langtang National Park. You can also create your own GeoJSON file for your AOI using sites like geojson.io



2.5.7 [Optional] Visually Verify your AOI

If you want to visually verify your AOI before proceeding, you may run the following cell, if you have the `geopandas` and `contextily` Python packages installed.

[21]:
```
try:
    import geopandas as gpd
    import contextily as ctx
except:
    print(
        "If you wish to visually verify your AOI, "
        "you must install the `geopandas` and `contextily` packages."
    )
else:
    aoi_gdf = gpd.read_file(aoi)
    aoi_epsg4326 = aoi_gdf.to_crs(epsg=4326)
    ax = aoi_epsg4326.plot(figsize=(10, 5), alpha=0.3, edgecolor="red")
    ctx.add_basemap(ax, crs=4326)
```



2.5.8 Submit a Job

When supplying input values for a GEDI subsetting job, to use the default value for a field (where indicated), use a dash ("-") as the input value.

- `aoi` (required): URL to a GeoJSON file representing your area of interest, as explained above.
- `doi`: Digital Object Identifier (DOI) of the GEDI collection to subset, or a logical name representing such a DOI. Valid logical names: L1B, L2A, L2B, L4A
- `columns`: Comma-separated list of column names to include in the output file.
- `query`: Query expression for subsetting the rows in the output file.
- `limit`: Maximum number of GEDI granule data files to download (among those that intersect the specified AOI). (Default: 10000)

It is recommended to use `maap-dps-worker-32gb` queues when submitting a job with a large aoi.

[22]:
```
inputs = dict(
    aoi=aoi,
    doi="L4A",
    lat="lat_lowestmode",
    lon="lon_lowestmode",
    beams="coverage",
    columns="agbd, agbd_se, sensitivity, geolocation/sensitivity_a2, elev_lowestmode",
    query="l2_quality_flag == 1 and l4_quality_flag == 1 and sensitivity > 0.95 and␣
↪`geolocation/sensitivity_a2` > 0.95",
    limit=10,
    temporal="-",
    output="gedi_subset.gpkg"
)

result = maap.submitJob(
    identifier="gedi-subset",
    algo_id="gedi-subset",
    version="0.6.0",
    queue="maap-dps-worker-32gb",
    username=username,
    **inputs,
)

job_id = result.id
job_id or result
```

[22]:
```
'72fca5ba-935a-49a2-802f-1dcfd3a5628c'
```

2.5.9 Get the Job's Output File

Now that the job has been submitted, we can use the `job_id` to check the job status until the job has been completed.

[23]:
```
from urllib.parse import urlparse


def job_status_for(job_id: str) -> str:
    return maap.getJobStatus(job_id)


def job_result_for(job_id: str) -> str:
    return maap.getJobResult(job_id)[0]


def to_job_output_dir(job_result_url: str) -> str:
    return f"/projects/my-private-bucket/{job_result_url.split(f'/{username}/')[1]}"
```

If you have installed the `backoff` Python package, running the following cell will automatically repeatedly check your job's status until the job has been completed. Otherwise, you will have to manually repeatedly rerun the following cell until the output is either `'Succeeded'` or `'Failed'`.

[24]:
```
try:
    import backoff
except:
    job_status = job_status_for(job_id)
else:
    # Check job status every 2 minutes
    @backoff.on_predicate(
        backoff.constant,
        lambda status: status not in ["Deleted", "Succeeded", "Failed"],
        interval=120,
    )
    def wait_for_job(job_id: str) -> str:
        return job_status_for(job_id)

    job_status = wait_for_job(job_id)

job_status
```

```
INFO:backoff:Backing off wait_for_job(...) for 0.9s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 18.1s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 49.5s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 6.8s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 42.4s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 26.7s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 86.6s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 117.0s (Accepted)
INFO:backoff:Backing off wait_for_job(...) for 17.9s (Running)
INFO:backoff:Backing off wait_for_job(...) for 95.7s (Running)
```

[24]:
```
'Succeeded'
```

[25]:
```
assert job_status == "Succeeded", (
```

(continues on next page)

(continued from previous page)

```
    job_result_for(job_id)
    if job_status == "Failed"
    else f"Job {job_id} has not yet completed ({job_status}). Rerun the prior cell."
)

output_url = job_result_for(job_id)
output_dir = to_job_output_dir(output_url)
output_file = f"{output_dir}/gedi_subset.gpkg"
print(f"Your subset results are in the file {output_file}")
```

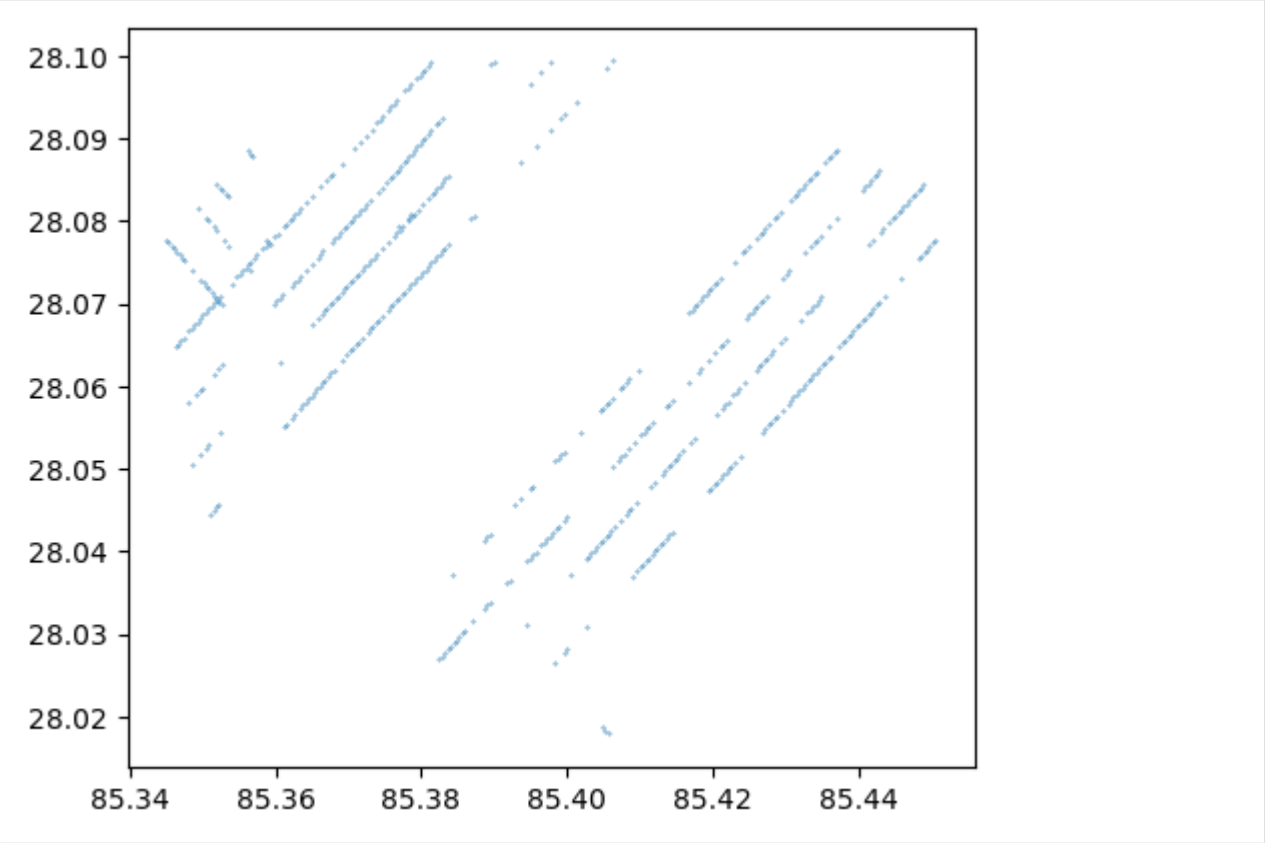
```
Your subset results are in the file /projects/my-private-bucket/dps_output/gedi-subset/0.
↪6.0/2023/06/27/20/05/21/764642/gedi_subset.gpkg
```

2.5.10 [Optional] Visually Verify the Results

If you installed the geopandas Python package, you can visually verify the output file by running the following cell.

[26]:
```
try:
    import geopandas as gpd
    import matplotlib.pyplot as plt
except:
    print(
        "If you wish to visually verify your output file, "
        "you must install the `geopandas` package."
    )
else:
    gedi_gdf = gpd.read_file(output_file)
    print(gedi_gdf.head())
    sensitivity_colors = plt.cm.get_cmap("viridis_r")
    gedi_gdf.plot(markersize = 0.1)
```

```
                                              filename  \
0  GEDI04_A_2020064181434_O06951_02_T04323_02_002...
1  GEDI04_A_2020064181434_O06951_02_T04323_02_002...
2  GEDI04_A_2020064181434_O06951_02_T04323_02_002...
3  GEDI04_A_2020064181434_O06951_02_T04323_02_002...
4  GEDI04_A_2020064181434_O06951_02_T04323_02_002...

   geolocation/sensitivity_a2  sensitivity  elev_lowestmode    agbd_se  \
0                    0.959091     0.959091      3256.992432  11.047880
1                    0.968629     0.968629      3282.666748  11.057747
2                    0.962079     0.962079      3314.994141  11.052886
3                    0.962610     0.962610      3351.686035  11.045983
4                    0.968436     0.968436      3436.938721  11.047858

         agbd                    geometry
0  169.741974  POINT (85.34628 28.06473)
1  235.977890  POINT (85.34667 28.06512)
2  189.141037  POINT (85.34705 28.06550)
3  180.132187  POINT (85.34742 28.06589)
4  191.731232  POINT (85.34817 28.06666)
```



2.5.11 Generate Contour Lines

Create a `lat`, `lon` mesh grid with elevation as a depth parameter. As shown in the plot above, the lines don't seem smooth. So we can apply `linear` or 'cubic` interpolation to smoothen those missing points.

[27]:
```
geometry = gedi_gdf["geometry"]
elevation=gedi_gdf["elev_lowestmode"]
```

[28]:
```
lon = geometry.x
lat = geometry.y
```

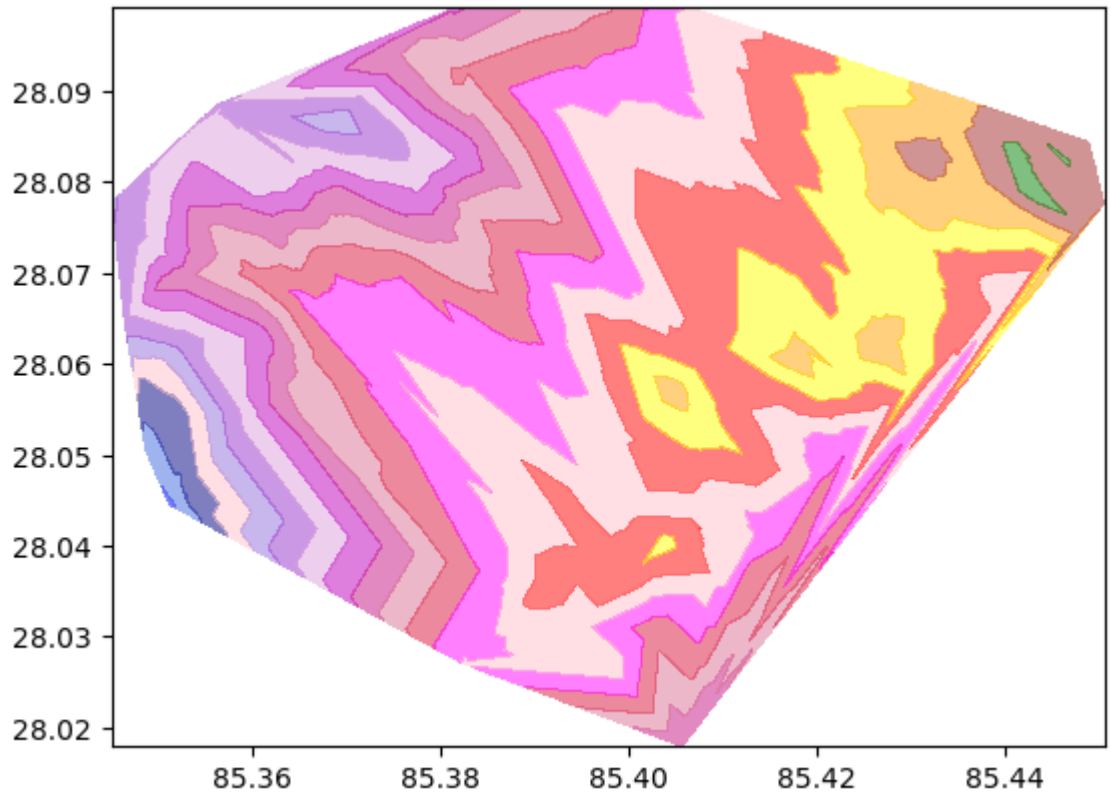
[29]:
```
import numpy as np

x=np.linspace(min(lon), max(lon), 1000)
y=np.linspace(min(lat), max(lat), 1000)
```

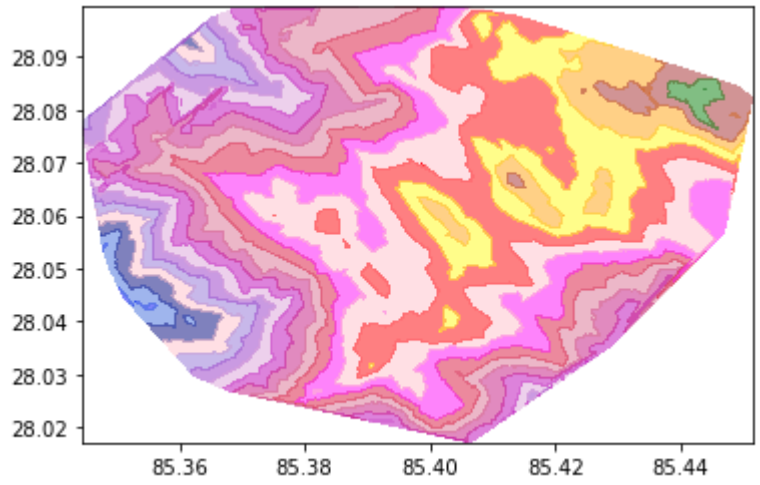
[30]:
```
from scipy.interpolate import griddata

x_mesh, y_mesh = np.meshgrid(x,y)
```

You may experiment with `nearest`, `linear`, and `cubic` interpolation methods to see which gives more smooth results.

[31]:
```
#grid the elevation
z_mesh = griddata((lon, lat), elevation, (x_mesh, y_mesh), method='linear')
```

[32]:
```
colors=['blue','royalblue', 'navy','pink',  'mediumpurple',  'darkorchid',  'plum',  'm',
↪ 'mediumvioletred', 'palevioletred', 'crimson',
        'magenta','pink','red','yellow','orange', 'brown','green', 'darkgreen']
levels=len(colors)
contourf = plt.contourf(x_mesh, y_mesh, z_mesh, levels, alpha=0.5,  colors=colors,␣
↪linestyles='None', vmin=elevation.min(), vmax=elevation.max())
```





Now we need to plot this contour into an interactive map for better visualization.

2.5.12 Plot the Contour Lines in Folium

You may need to install `geojsoncontour`, `mapclassify`, and `folium`, if you don't already have them installed. We need to convert this `contourf` into geoJSON format.

[33]:
```
import folium
from folium import plugins
import branca
import geojsoncontour
```

[34]:
```
geojson = geojsoncontour.contourf_to_geojson(
    contourf=contourf,
    min_angle_deg=3.0,
    ndigits=5,
    stroke_width=1,
    unit='ft',
    fill_opacity=0.1,
)
```

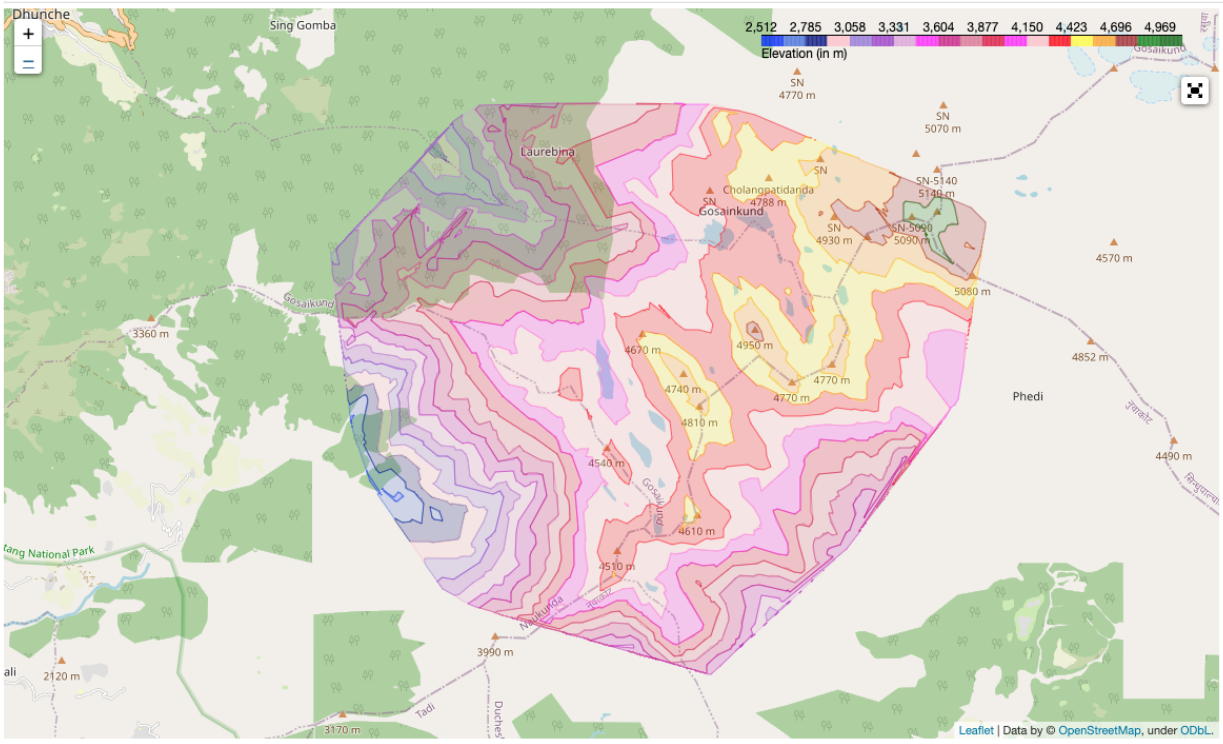
[35]:
```
#create map view
m = folium.Map([lat.mean(), lon.mean()], zoom_start=12, tiles="OpenStreetMap")

folium.GeoJson(
    geojson,
    style_function=lambda x:{
        'color': x['properties']['stroke'],
        'weight':    x['properties']['stroke-width'],
        'fillColor': x['properties']['fill'],
        'opacity':   0.5,
    }
).add_to(m)

cm = branca.colormap.LinearColormap(colors, vmin=elevation.min(), vmax=elevation.max()).
↪to_step(levels)
cm.caption='Elevation (in m)'
m.add_child(cm)

#legend
plugins.Fullscreen(position='topright', force_separate_button=True).add_to(m)
```

[35]:
```
<folium.plugins.fullscreen.Fullscreen at 0x7f356182c4c0>
```

[36]:
```
m
```

[36]:
```
<folium.folium.Map at 0x7f356182c670>
```

Now you have an interactive visualization of a contour plot.



2.6 GEDI_L4B Search and Visualize

Authors: Nikita Susan (UAH), Aimee Barciauskas (DevSeed), Sumant Jha (MSFC/USRA), Alex Mandel (DevSeed)

Date: April 7, 2023

Description: In this example, we demonstrate how to access the GEDI L4B collection and granule data on the MAAP ADE, and then visualize the data using matplotlib.

2.6.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.6.2 About the Data

GEDI L4B Gridded Aboveground Biomass Density, Version 2

This dataset provides Global Ecosystem Dynamics Investigation (GEDI) Level 4 (L4) data, which has the purpose of providing mean aboveground biomass density (AGBD) and consists of the GEDI_L4A and GEDI_L4B collections. GEDI_L4B uses a sample present within each 1km cell to statistically infer mean AGBD. GEDI is attached to the International Space Station (ISS) and collects data globally between 51.6° N and 51.6° S latitudes at the highest resolution and densest sampling of any light detection and ranging (lidar) instrument in orbit to date; specifically, GEDI L4B data has a spatial resolution of 1km. (Source: GEDI_L4B Version 2 User Guide)

2.6.3 Additional Resources

- GEDI_L4B Version 2 Dataset Landing Page
- The GEDI Website
- Earthdata Search

2.6.4 Importing Packages

Within your Jupyter Notebook, start by importing the **maap package**. Then invoke the **MAAP** constructor, setting the maap_host argument to **'api.maap-project.org'**.

[22]:
```
from maap.maap import MAAP
from matplotlib import pyplot
import os
import pprint
import rasterio
import boto3

maap = MAAP(maap_host="api.maap-project.org")
```

2.6.5 Search for the Collection and Associated Granules

Now, we will search for the collection using the collection short name:

[23]:
```
collection = maap.searchCollection(cmr_host='cmr.earthdata.nasa.gov', short_name="GEDI_
↪L4B_Gridded_Biomass_2017", limit=100)
print(collection)
```

```
[{'concept-id': 'C2244602422-ORNL_CLOUD', 'revision-id': '7', 'format': 'application/
↪echo10+xml', 'Collection': {'ShortName': 'GEDI_L4B_Gridded_Biomass_2017', 'VersionId':
↪'2', 'InsertTime': '2022-03-29T00:00:00Z', 'LastUpdate': '2023-06-12T20:25:17Z',
↪'LongName': 'GEDI L4B Gridded Aboveground Biomass Density, Version 2', 'DataSetId':
↪'GEDI L4B Gridded Aboveground Biomass Density, Version 2', 'Description': "This Global␣
↪Ecosystem Dynamics Investigation (GEDI) L4B product provides 1 km x 1 km (1 km, ␣
↪hereafter) estimates of mean aboveground biomass density (AGBD) based on observations␣
↪from mission week 19 starting on 2019-04-18 to mission week 138 ending on 2021-08-04.␣
↪The GEDI L4A Footprint Biomass product converts each high-quality waveform to an AGBD␣
↪prediction, and the L4B product uses the sample present within the borders of each 1␣
↪km cell to statistically infer mean AGBD. The gridding procedure is described in the␣
↪GEDI L4B Algorithm Theoretical Basis Document (ATBD). Patterson et al. (2019)␣
↪describes the hybrid model-based mode of inference used in the L4B product.␣
↪Corresponding 1 km estimates of the standard error of the mean are also provided in␣
↪the L4B product. Uncertainty is due to both GEDI's sampling of the 1 km area (as␣
↪opposed to making wall-to-wall observations) and the fact that L4A biomass values are␣
↪modeled in a process subject to error instead of measured in a process that may be␣
↪assumed to be error-free.", 'DOI': {'DOI': '10.3334/ORNLDAAC/2017', 'Authority':
↪'https://doi.org'}, 'Orderable': 'false', 'Visible': 'true', 'RevisionDate': '2022-03-
↪29T00:00:00Z', 'ProcessingLevelId': '4', 'ProcessingLevelDescription': 'model products
↪', 'ArchiveCenter': 'ORNL_DAAC', 'CitationForExternalPublication': 'Dubayah, R.O., J.␣
↪Armston, S.P. Healey, Z. Yang, P.L. Patterson, S. Saarela, G. Stahl, L. Duncanson, and␣
↪J.R. Kellner. 2022. GEDI L4B Gridded Aboveground Biomass Density, Version 2. ORNL DAAC,
```

(continues on next page)

(continued from previous page)


```
↪ Oak Ridge, Tennessee, USA. https://doi.org/10.3334/ORNLDAAC/2017', 'CollectionState':
↪'COMPLETE', 'MaintenanceAndUpdateFrequency': 'As needed', 'UseConstraints': {
↪'LicenseURL': {'URL': 'https://science.nasa.gov/earth-science/earth-science-data/data-
↪information-policy', 'Description': 'License URL for data use policy', 'Type': 'Data␣
↪Use Policy', 'MimeType': 'text/html'}}, 'Price': '0', 'DataFormat': 'GeoTIFF',
↪'SpatialKeywords': {'Keyword': 'GLOBAL LAND'}, 'Temporal': {'RangeDateTime': {
↪'BeginningDateTime': '2019-04-18T00:00:00Z', 'EndingDateTime': '2021-08-04T23:59:59Z'}}
↪, 'Contacts': {'Contact': {'Role': 'ARCHIVER', 'OrganizationName': 'ORNL_DAAC',
↪'OrganizationAddresses': {'Address': {'StreetAddress': 'ORNL DAAC User Services Office,
↪ P.O. Box 2008, MS 6407, Oak Ridge National Laboratory', 'City': 'Oak Ridge',
↪'StateProvince': 'Tennessee', 'PostalCode': '37831-6407', 'Country': 'USA'}},
↪'OrganizationPhones': {'Phone': {'Number': '(865) 241-3952', 'Type': 'Direct Line'}},
↪'OrganizationEmails': {'Email': 'uso@daac.ornl.gov'}}}, 'ScienceKeywords': {
↪'ScienceKeyword': [{'CategoryKeyword': 'EARTH SCIENCE', 'TopicKeyword': 'BIOSPHERE',
↪'TermKeyword': 'ECOSYSTEMS', 'VariableLevel1Keyword': {'Value': 'TERRESTRIAL ECOSYSTEMS
↪'}}, {'CategoryKeyword': 'EARTH SCIENCE', 'TopicKeyword': 'BIOSPHERE', 'TermKeyword':
↪'VEGETATION', 'VariableLevel1Keyword': {'Value': 'BIOMASS'}}, {'CategoryKeyword':
↪'EARTH SCIENCE', 'TopicKeyword': 'SPECTRAL/ENGINEERING', 'TermKeyword': 'LIDAR',
↪'VariableLevel1Keyword': {'Value': 'LIDAR WAVEFORM'}}]}, 'Platforms': {'Platform': {
↪'ShortName': 'ISS', 'LongName': 'MUSES', 'Type': 'Space Stations/Crewed Spacecraft',
↪'Instruments': {'Instrument': {'ShortName': 'GEDI', 'LongName': 'Global Ecosystem␣
↪Dynamics Investigation'}}}}, 'Campaigns': {'Campaign': {'ShortName': 'GEDI', 'LongName
↪': 'Global Ecosystem Dynamics Investigation'}}, 'OnlineAccessURLs': {'OnlineAccessURL':
↪ {'URL': 'https://daac.ornl.gov/gedi/GEDI_L4B_Gridded_Biomass/', 'URLDescription':
↪'This link allows direct data access via Earthdata login'}}, 'OnlineResources': {
↪'OnlineResource': [{'URL': 'https://daac.ornl.gov/GEDI/guides/GEDI_L4B_Gridded_Biomass.
↪html', 'Description': 'ORNL DAAC Data Set Documentation', 'Type': "USER'S GUIDE"}, {
↪'URL': 'https://doi.org/10.3334/ORNLDAAC/2017', 'Description': 'Data set Landing Page␣
↪DOI URL', 'Type': 'DATA SET LANDING PAGE'}, {'URL': 'https://data.ornldaac.earthdata.
↪nasa.gov/public/gedi/GEDI_L4B_Gridded_Biomass/comp/GEDI_L4B_ATBD_v1.0.pdf',
↪'Description': 'GEDI L4B Gridded Aboveground Biomass Density, Version 2: GEDI_L4B_ATBD_
↪v1.0.pdf', 'Type': 'GENERAL DOCUMENTATION'}, {'URL': 'https://data.ornldaac.earthdata.
↪nasa.gov/public/gedi/GEDI_L4B_Gridded_Biomass/comp/GEDI_L4B_Gridded_Biomass.pdf',
↪'Description': 'GEDI L4B Gridded Aboveground Biomass Density, Version 2: GEDI_L4B_
↪Gridded_Biomass.pdf', 'Type': 'GENERAL DOCUMENTATION'}, {'URL': 'https://daac.ornl.gov/
↪GEDI/guides/GEDI_L4B_Gridded_Biomass_Fig1.png', 'Description': 'Gridded mean␣
↪aboveground biomass density (top) and standard error of the mean (bottom).', 'Type':
↪'GET RELATED VISUALIZATION', 'MimeType': 'image/png'}, {'URL': 'https://gedi.umd.edu',
↪'Description': 'GEDI Project Site', 'Type': 'PROJECT HOME PAGE', 'MimeType': 'text/html
↪'}, {'URL': 'https://webmap.ornl.gov/wcsdown/dataset.jsp?ds_id=2017', 'Description':
↪'Web Coverage Service for this collection.', 'Type': 'WEB COVERAGE SERVICE (WCS)',
↪'MimeType': 'application/gml+xml'}]}, 'Spatial': {'SpatialCoverageType': 'HORIZONTAL',
↪'HorizontalSpatialDomain': {'Geometry': {'CoordinateSystem': 'CARTESIAN',
↪'BoundingRectangle': {'WestBoundingCoordinate': '-180.00', 'NorthBoundingCoordinate':
↪'52.00', 'EastBoundingCoordinate': '180.00', 'SouthBoundingCoordinate': '-52.00'}}},
↪'GranuleSpatialRepresentation': 'CARTESIAN'}, 'DirectDistributionInformation': {'Region
↪': 'us-west-2', 'S3BucketAndObjectPrefixName': 's3://ornl-cumulus-prod-protected/gedi/
↪GEDI_L4B_Gridded_Biomass/', 'S3CredentialsAPIEndpoint': 'https://data.ornldaac.
↪earthdata.nasa.gov/s3credentials', 'S3CredentialsAPIDocumentationURL': 'https://data.
↪ornldaac.earthdata.nasa.gov/s3credentialsREADME'}}}]
```


Next, we can search for granules using the searchGranule function and the concept ID from our collection search above:

[24]:
```
COLLECTIONID = collection[0]['concept-id']
results = maap.searchGranule(cmr_host='cmr.earthdata.nasa.gov',concept_id=COLLECTIONID)
↪# COLLECTIONID 'C2244602422-ORNL_CLOUD'
print(f'Got {len(results)} results')
results[0]['Granule']
```

```
Got 10 results
```

[24]:
```
{'GranuleUR': 'GEDI_L4B_Gridded_Biomass.GEDI04_B_MW019MW138_02_002_05_R01000M_PS.tif',
 'InsertTime': '2022-03-29T00:00:00Z',
 'LastUpdate': '2023-04-10T21:58:24Z',
 'Collection': {'ShortName': 'GEDI_L4B_Gridded_Biomass_2017',
  'VersionId': '2'},
 'DataGranule': {'DataGranuleSizeInBytes': '20103343',
  'SizeMBDataGranule': '20.103343',
  'Checksum': {'Value': '025a141348906d5e612262218c496a2d468446ca30875439be6651d851bfbe23
↪',
   'Algorithm': 'SHA-256'},
  'DayNightFlag': 'BOTH',
  'ProductionDateTime': '2022-03-29T00:00:00Z'},
 'Temporal': {'RangeDateTime': {'BeginningDateTime': '2019-04-18T00:00:00Z',
   'EndingDateTime': '2021-08-04T23:59:59Z'}},
 'Spatial': {'HorizontalSpatialDomain': {'Geometry': {'BoundingRectangle': {
↪'WestBoundingCoordinate': '-180',
     'NorthBoundingCoordinate': '52',
     'EastBoundingCoordinate': '180',
     'SouthBoundingCoordinate': '-52'}}}},
 'MeasuredParameters': {'MeasuredParameter': [{'ParameterName': 'LIDAR WAVEFORM'},
   {'ParameterName': 'BIOMASS'},
   {'ParameterName': 'TERRESTRIAL ECOSYSTEMS'}]},
 'Platforms': {'Platform': {'ShortName': 'ISS',
   'Instruments': {'Instrument': {'ShortName': 'GEDI'}}}},
 'Campaigns': {'Campaign': {'ShortName': 'GEDI'}},
 'Price': '0',
 'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://data.ornldaac.earthdata.nasa.
↪gov/protected/gedi/GEDI_L4B_Gridded_Biomass/data/GEDI04_B_MW019MW138_02_002_05_R01000M_
↪PS.tif',
    'URLDescription': 'Download GEDI04_B_MW019MW138_02_002_05_R01000M_PS.tif',
    'MimeType': 'image/tiff'},
   {'URL': 'https://data.ornldaac.earthdata.nasa.gov/public/gedi/GEDI_L4B_Gridded_
↪Biomass/data/GEDI04_B_MW019MW138_02_002_05_R01000M_PS.tif.sha256',
    'URLDescription': 'Download GEDI04_B_MW019MW138_02_002_05_R01000M_PS.tif.sha256'},
   {'URL': 's3://ornl-cumulus-prod-protected/gedi/GEDI_L4B_Gridded_Biomass/data/GEDI04_B_
↪MW019MW138_02_002_05_R01000M_PS.tif',
    'URLDescription': 'This link provides direct download access via S3 to the granule',
    'MimeType': 'image/tiff'}]},
 'OnlineResources': {'OnlineResource': [{'URL': 'https://daac.ornl.gov/GEDI/guides/GEDI_
↪L4B_Gridded_Biomass.html',
    'Description': 'ORNL DAAC Data Set Documentation',
    'Type': "USER'S GUIDE"},
   {'URL': 'https://doi.org/10.3334/ORNLDAAC/2017',
    'Description': 'Data set Landing Page DOI URL',
    'Type': 'DATA SET LANDING PAGE'},
   {'URL': 'https://daac.ornl.gov/daacdata/gedi/GEDI_L4B_Gridded_Biomass/comp/GEDI_L4B_
```

(continues on next page)

(continued from previous page)

```
↪ATBD_v1.0.pdf',
    'Description': 'Data Set Documentation',
    'Type': 'GENERAL DOCUMENTATION'},
   {'URL': 'https://daac.ornl.gov/daacdata/gedi/GEDI_L4B_Gridded_Biomass/comp/GEDI_L4B_
↪Gridded_Biomass.pdf',
    'Description': 'Data Set Documentation',
    'Type': 'GENERAL DOCUMENTATION'},
   {'URL': 'https://webmap.ornl.gov/sdat/pimg/2017_9.png',
    'Description': 'GEDI L4B Gridded Prediction Stratum, Version 2, Mission Weeks 19-138
↪',
    'Type': 'BROWSE',
    'MimeType': 'image/png'},
   {'URL': 'https://data.ornldaac.earthdata.nasa.gov/s3credentials',
    'Description': 'api endpoint to retrieve temporary credentials valid for same-region␣
↪direct s3 access',
    'Type': 'VIEW RELATED INFORMATION'}]},
 'Orderable': 'false',
 'DataFormat': 'COG'}
```

2.6.6 Accessing and Downloading the Granule from ORNL DAAC S3

Before downloading, we'll get the collection and file name:

[25]:
```
granule_ur=results[0]['Granule']['GranuleUR'].split(".")
collection_name=granule_ur[0]
file_name=granule_ur[1]

print(collection_name)
print(file_name)
```

```
GEDI_L4B_Gridded_Biomass
GEDI04_B_MW019MW138_02_002_05_R01000M_PS
```

Now we'll proceed to get tempory s3 credentials, and then download the tif file to our workspace:

[26]:
```
def get_s3_creds(url):
    return maap.aws.earthdata_s3_credentials(url)

def get_s3_client(s3_cred_endpoint):
    creds=get_s3_creds(s3_cred_endpoint)
    boto3_session = boto3.Session(
            aws_access_key_id=creds['accessKeyId'],
            aws_secret_access_key=creds['secretAccessKey'],
            aws_session_token=creds['sessionToken']
    )
    return boto3_session.client("s3")

def download_s3_file(s3, bucket, collection_name, file_name):
    os.makedirs("/projects/gedi_l4b", exist_ok=True) # create directories, as necessary
    download_path=f"/projects/gedi_l4b/{file_name}.tif"
    s3.download_file(bucket, f"gedi/{collection_name}/data/{file_name}.tif", download_
```

(continues on next page)

(continued from previous page)

```
↪path)
    return download_path
```

[27]:
```
s3_cred_endpoint= 'https://data.ornldaac.earthdata.nasa.gov/s3credentials'
s3=get_s3_client(s3_cred_endpoint)
```

[28]:
```
bucket="ornl-cumulus-prod-protected"
download_path=download_s3_file(s3, bucket, collection_name, file_name)
download_path
```

[28]:
```
'/projects/gedi_l4b/GEDI04_B_MW019MW138_02_002_05_R01000M_PS.tif'
```

Open the local file using rasterio, and print the shape of the data to verify if the file was read properly:

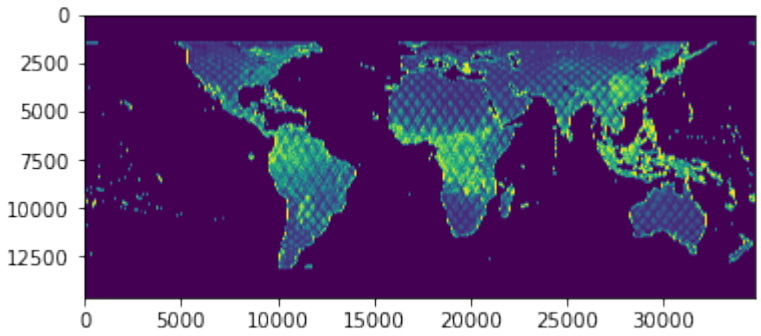
[29]:
```
src = rasterio.open(download_path)
data = src.read(1)

print(data.shape)
```

```
(14616, 34704)
```

2.6.7 Plot the Data

Finally, we'll use matplotlib to visualize our .tif file:

[30]:
```
pyplot.imshow(data)
pyplot.show()
```



2.7 ICESat-02 ATL03 Subset and Visualize

Authors: Sumant Jha (MSFC/USRA), Samuel Ayers (UAH), Alex Mandel (DevSeed), Aimee Barciauskas (DevSeed)

Date: March 6, 2023

Description: In this tutorial, we will search for ATL03 data within the NASA CMR. We will then read and visualize the data structure of a granule, create a subset and data frames, and visualize the photon heights.

2.7.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.7.2 About the Data

ATLAS/ICESat-2 L2A Global Geolocated Photon Data, Version 5

"This data set [ATL03] contains height above the WGS 84 ellipsoid (ITRF2014 reference frame), latitude, longitude, and time for all photons downlinked by the Advanced Topographic Laser Altimeter System (ATLAS) instrument on board the Ice, Cloud and land Elevation Satellite-2 (ICESat-2) observatory. The ATL03 product was designed to be a single source for all photon data and ancillary information needed by higher-level ATLAS/ICESat-2 products. As such, it also includes spacecraft and instrument parameters and ancillary data not explicitly required for ATL03." (Source: ATL03 Dataset Landing Page)

2.7.3 Additional Resources

- ATL03 Version 5 User Guide
- Earthdata Search

2.7.4 Importing and Installing Packages

Required packages:

You will need to install the following required packages if not already present in your working environment: maap-py, pandas, geopandas, folium, shapely, h5glance, h5py

[1]:
```
# ! pip install geopandas
# ! pip install folium
# ! pip install h5glance
```

[2]:
```
# Import the MAAP package
from maap.maap import MAAP

# Invoke the MAAP constructor using the maap_host argument
maap = MAAP(maap_host='api.maap-project.org')

# Import pandas dataframe
import pandas as pd

# Import libraries needed for visualizing data spatial extent
import geopandas as gpd
import folium
from shapely.geometry import Polygon,Point

# Import H5glance to interactively explore H5 file in notebook
from h5glance import H5Glance
```

(continues on next page)

(continued from previous page)

```
# Import H5py to read h5 file
import h5py

# Import os to create a new directory
import os
```

2.7.5 Decide on a Subset of ATL03 Data

First, we will create a subset using a spatial extent and date range before visualizaing using folium. For this tutorial, we will focus on a very small area over Yosemite National Park and use a temporal range of one day.

[3]:
```
# Create a variable for short name of ATL03 data
short_name = 'ATL03'

# Create Latitude, Longitude list.
lat_coords = [37.700057,37.700057,37.758166,37.758166,37.700057]
lon_coords = [-119.680359,-119.680359,-119.538910,-119.538910,-119.680359]

# Create bounding box
spatial_extent = [lon_coords[0],lat_coords[0],lon_coords[2],lat_coords[2]]

# Reformat bounding box to work with NASA CMR API
spatial_extent = ','.join(str(coords) for coords in spatial_extent)

#Provide date range. It is just 1 day.
date_range = ['2021-02-02','2022-02-03']

# For folium purpose, provide the map center
map_center = [37.729139,-119.609670]

# Convert to AOI for visualizaton with folium
polygon_geom = Polygon(zip(lon_coords, lat_coords))

# Provide relevant Coordinate Reference System
crs = 'epsg:4326'

# Convert to Geodataframe and back to list in that specific reference system
AOI = gpd.GeoDataFrame(index=[0], crs=crs, geometry=[polygon_geom])
AOI_bbox = AOI.bounds.iloc[0].to_list()
```

[4]:
```
# Visualize the spatial extent using folium.
m = folium.Map(map_center, zoom_start=12, tiles='OpenStreetMap')
folium.GeoJson(AOI).add_to(m)
folium.LatLngPopup().add_to(m)
m
```

[4]:
```
<folium.folium.Map at 0x7f50c1cffb50>
```

2.7.6 Search for Granules

We will now search for granules in NASA's CMR given our spatial extent and date range, then print the number of granules available.

[5]:
```
# Provide the name of cmr host to use with maap py
nasa_cmr_host = "cmr.earthdata.nasa.gov"
# Search granule using maap-py using subset criteria identified earlier.
data = maap.searchGranule(cmr_host=nasa_cmr_host,short_name = short_name, bounding_box =␣
↪spatial_extent,temporal= date_range,limit=1000)
# Check to see if the granule search was successfull in finding data within spatial and␣
↪temporal extent
print(len(data))
```

```
104
```

Let's create a new data directory and use maap-py's getData functon to extract the first H5 file. We will then print the name of the extracted file.

[6]:
```
# set data directory
dataDir = "./data"

# check if directory exists -> if directory doesn't exist, directory is created
if not os.path.exists(dataDir):
    os.mkdir(dataDir)

# download and extract the resource
ice_data = data[0].getData(dataDir)
print(ice_data)
```

```
./data/ATL03_20210202191800_06231006_005_01.h5
```

2.7.7 Read the H5 File

Let's read in the H5 file to understand the data structure. There are two ways to do this. We can just list keys and then go forward exploring each key one by one…

[7]:
```
# Open the H5 file and list the keys
ice_file = h5py.File(ice_data,'r')
list(ice_file.keys())
```

[7]:
```
['METADATA',
 'ancillary_data',
 'atlas_impulse_response',
 'ds_surf_type',
 'ds_xyz',
 'gt1l',
 'gt1r',
 'gt2l',
 'gt2r',
 'gt3l',
 'gt3r',
 'orbit_info',
 'quality_assessment']
```

... or use the h5glance package to interactively list various keys, sub-keys and variables.

H5Glance lists all the keys and sub-keys and allows for the copying of path variables on the fly, all from within the Jupyter Notebook. Note: In the web version of this notebook which is shown here, not all sub-fields will be listed. But they can be accessed when using a Jupyter Notebook running on ADE or local machines.

[8]:
```
H5Glance(ice_file)
```

[8]:
```
./data/ATL03_20210202191800_06231006_005_01.h5/ (47 attributes)
├METADATA   (9 children) (3 attributes)
├ancillary_data    (34 children) (2 attributes)
├atlas_impulse_response   (2 children) (1 attributes)
├ds_surf_type     [int32: 5] (12 attributes)
├ds_xyz    [int32: 3] (12 attributes)
├gt1l       (5 children) (7 attributes)
├gt1r       (5 children) (7 attributes)
├gt2l       (5 children) (7 attributes)
├gt2r       (5 children) (7 attributes)
├gt3l       (5 children) (7 attributes)
├gt3r       (5 children) (7 attributes)
├orbit_info        (7 children) (2 attributes)
└quality_assessment        (9 children) (1 attributes)
```

2.7.8 Subset the Data by Required Columns

In this case, we will need Latitude, Longitude, Photon Height and Along Track Distance. We are using the copied path from the data tree generated by H5Glance above.

Use h5py to read file and variables from the copied path.

[9]:
```
with h5py.File(ice_data,'r') as f:
    gt1l_lat = f['/gt1l/heights/lat_ph'][:]
    gt1l_lon = f['/gt1l/heights/lon_ph'][:]
    gt1l_height = f['/gt1l/heights/h_ph'][:]
    gt1l_dist_ph = f['/gt1l/heights/dist_ph_along'][:]
```

2.7.9 Show the Subset Data in a Dataframe

1. By using the Pandas module:

[10]:
```
# Write latitude, longitude, photon height and along track distance for gt1l to a␣
↪dataframe
gt1l_df = pd.DataFrame({'Latitude': gt1l_lat, 'Longitude': gt1l_lon, 'Photon_Height':␣
↪gt1l_height,'Along_track_distance':gt1l_dist_ph})
gt1l_df
```

[10]:

	Latitude	Longitude	Photon_Height	Along_track_distance
0	59.482065	-115.906874	611.887878	16.894447
1	59.482065	-115.906877	580.084106	16.980614
2	59.482064	-115.906882	527.962097	17.122099
3	59.482063	-115.906885	491.036133	17.222527
4	59.482059	-115.906875	616.919922	17.593958
...	...	...	...	...

(continues on next page)

(continued from previous page)

```
36561868  33.011390 -119.752303      69.065865              4.154013
36561869  33.011382 -119.752316     -16.501354              5.156162
36561870  33.011380 -119.752329    -108.562981              5.470425
36561871  33.011376 -119.752317     -17.947205              5.871534
36561872  33.011375 -119.752319     -37.489002              5.938596

[36561873 rows x 4 columns]
```

2. By using the Geopandas module:

Create geopandas dataframe with a column for point locations in the geometry column, and other relevant variables.

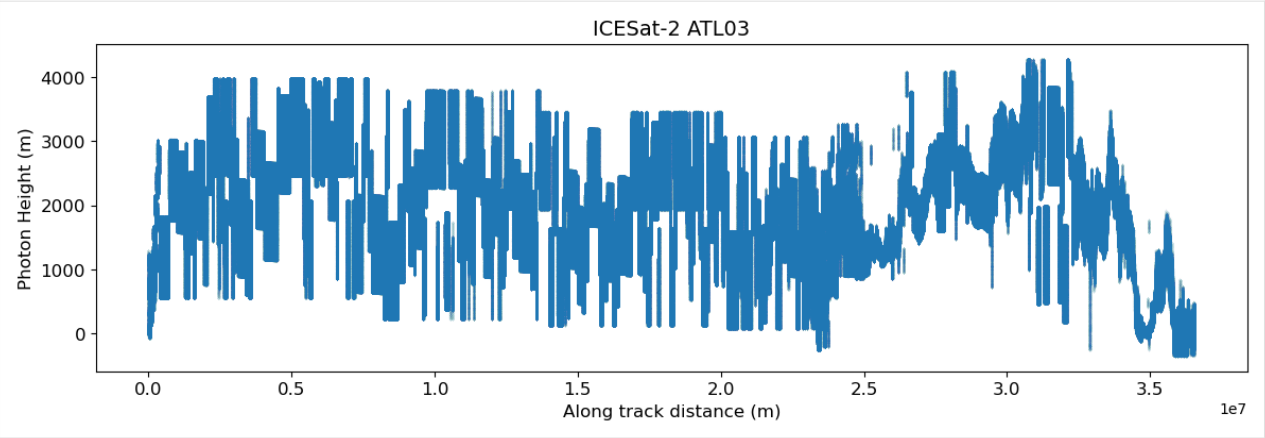
[11]:
```
geometry = gpd.points_from_xy(gt1l_lon, gt1l_lat)
data = {'Latitude': gt1l_lat, 'Longitude': gt1l_lon, 'Photon_Height': gt1l_height,'Along_
↪track_distance':gt1l_dist_ph}
gdf = gpd.GeoDataFrame(data,geometry=geometry, crs='EPSG:4326')

# View the resulting geopandas dataframe
print(gdf.head())
```

```
    Latitude   Longitude  Photon_Height  Along_track_distance  \
0  59.482065 -115.906874     611.887878             16.894447
1  59.482065 -115.906877     580.084106             16.980614
2  59.482064 -115.906882     527.962097             17.122099
3  59.482063 -115.906885     491.036133             17.222527
4  59.482059 -115.906875     616.919922             17.593958

                     geometry
0  POINT (-115.90687 59.48207)
1  POINT (-115.90688 59.48206)
2  POINT (-115.90688 59.48206)
3  POINT (-115.90689 59.48206)
4  POINT (-115.90687 59.48206)
```

2.7.10 Visualize Photon Heights

Finally, we'll visualize the photon heights with respect to along track distance for this H5 file using inputs from the geodataframe:

[12]:
```
import matplotlib.pyplot as plt
fig, ax = plt.subplots(figsize=(14, 4))
gdf.Photon_Height.plot(ax=ax, ls='', marker='.', ms=0.01)
ax.set_xlabel('Along track distance (m)', fontsize=12);
ax.set_ylabel('Photon Height (m)', fontsize=12)
ax.set_title('ICESat-2 ATL03', fontsize=14)
ax.tick_params(axis='both', which='major', labelsize=12)
```



2.8 NISAR Access and Visualize

Authors: Sumant Jha (MSFC/USRA), Emile Tenezakis (DevSeed), Alex Mandel (DevSeed), Samuel Ayers (UAH)

Date: March 9, 2023

Description: In the following tutorial, we are going to look at how to access, read, and visualize simulated-NISAR data which is available from the NISAR mission. This mission is expected to launch in 2024.

2.8.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.8.2 About the Data

Unmanned Aerial Vehicle Synthetic Aperture Radar (UAVSAR)

This tutorial currently uses UAVSAR data, which has the purpose of mapping crustal deformations associated with natural hazards. UAVSAR data can also include topographic data which are derived from phase measurements. If two or more passes are made, UAVSAR data can be used to detect changes on the surface of the Earth, and is overall a highly useful tool in monitoring natural hazards and disasters. (Source: ESA eoPortal - UAVSAR)

Note: UAVSAR is a predecessor to NASA-ISRO Synthetic Aperture Radar (NISAR), which is a mission that will likely be launched in 2024. At this time we are using UAVSAR data to simulate the data that we will get from the NISAR satellite.

2.8.3 Additional Resources

- Alaska Satellite Facility (ASF) UAVSAR
- JPL - NISAR Mission
- JPL - SAR Overview
- ASF - SAR Media Resources

2.8.4 Import and Install Packages

Before we start any work, we will need to prepare our Jupyter workspace with the necessary python packages. In this tutorial we rely on - h5py, numpy, glob, os, h5glance, osgeo, and matplotlib.

You might have some or all packages already installed. Follow the instructions below to check and install the packages required for this tutorial. The block below will check if your packages are available and install them if they are not.

[1]:
```
# %pip install h5py glob2 h5glance gdal matplotlib pystac-client
```

We also require that you have the gdal package installed out of the Jupyter environment to use: gdal_translate and gdalinfo. If on Mac OS, you might need to install the xcode package before being able to install gdal.

To install gdal outside the Jupyter environment, please follow the instructions here as it will vary by the OS you are using: https://gdal.org/download.html

Import installed packages to read the NISAR h5 files and look at the data structure.

[2]:
```
import h5py
import glob
import os
from h5glance import H5Glance
from pystac_client import Client
```

2.8.5 Download NISAR Data

NISAR data can be downloaded from https://uavsar.jpl.nasa.gov/cgi-bin/data.pl. At the time of this tutorial, there are 21033 NISAR products available through the above link. There are a few ways to download data that we can use. We can directly download an SLC file or we can download an H5 file and translate it to SLC to work with this tutorial. In this tutorial, we demonstrate how to get an H5 file, translate it to SLC and use it. Users can alternatively directly download SLC files.

We can browse the 129 NISAR mode products in the MAAP STAC catalog and retrieve its asset HREFs for download.

[3]:
```
cat = Client.open("https://stac.maap-project.org/")

items = list(cat.search(collections="nisar-sim", max_items=10).items())
```

[4]:
```
item_asset = items[0].assets["CX_129.h5"].href
```

Before downloading, we'll create a new data directory to download the file into.

[5]:
```
# set data directory
dataDir = "./data"
```

(continues on next page)

(continued from previous page)

```
# check if directory exists -> if directory doesn't exist, directory is created
if not os.path.exists(dataDir):
    os.mkdir(dataDir)
```

[6]:
```
# download the file using wget
!wget -P {dataDir} {item_asset}
```

```
--2023-09-18 14:52:57--  https://downloaduav.jpl.nasa.gov/Release2z/Haywrd_14501_21043_
↪012_210602_L090_CX_02/Haywrd_14501_21043_012_210602_L090_CX_129_02.h5
Resolving downloaduav.jpl.nasa.gov (downloaduav.jpl.nasa.gov)... 137.78.249.121
Connecting to downloaduav.jpl.nasa.gov (downloaduav.jpl.nasa.gov)|137.78.249.121|:443...␣
↪connected.
HTTP request sent, awaiting response... 301 Moved Permanently
Location: https://uavsar.jpl.nasa.gov/Release2z/Haywrd_14501_21043_012_210602_L090_CX_02/
↪Haywrd_14501_21043_012_210602_L090_CX_129_02.h5 [following]
--2023-09-18 14:52:57--  https://uavsar.jpl.nasa.gov/Release2z/Haywrd_14501_21043_012_
↪210602_L090_CX_02/Haywrd_14501_21043_012_210602_L090_CX_129_02.h5
Resolving uavsar.jpl.nasa.gov (uavsar.jpl.nasa.gov)... 137.78.249.121
Connecting to uavsar.jpl.nasa.gov (uavsar.jpl.nasa.gov)|137.78.249.121|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 1750205900 (1.6G)
Saving to: ‘./data/Haywrd_14501_21043_012_210602_L090_CX_129_02.h5.2’

Haywrd_14501_21043_ 100%[===================>]   1.63G  53.5MB/s    in 35s

2023-09-18 14:53:40 (48.3 MB/s) - ‘./data/Haywrd_14501_21043_012_210602_L090_CX_129_02.
↪h5.2’ saved [1750205900/1750205900]
```

The file should now appear in the directory we created.

If you downloaded multiple H5 files, initialize an empty variable to store the names of all H5 files in the directory.

[7]:
```
nisar_list = []
```

[8]:
```
for file in glob.glob("./data*/*.h5"):
    nisar_list.append(file)
    print(file)
```

```
./data/Haywrd_14501_21043_012_210602_L090_CX_129_02.h5
```

2.8.6 Explore the Data

Explore the structure of NISAR *.H5 file using H5glance.

[10]:
```
H5Glance(nisar_list[0])
```

[10]:
```
./data/Haywrd_14501_21043_012_210602_L090_CX_129_02.h5/ (6 attributes)
└science    (1 children)
```

2.8.7 Convert the Data

Use gdal to convert HH polarization for frequencyA to ENVI SLC format and visualize it. (More example notebooks are provided here: https://github.com/isce-framework/sds-ondemand)

Before we go ahead and convert the HH polarization, check out the metadata information associated with it.

[11]:
```
path = f"HDF5:{nisar_list[0]}://science/LSAR/SLC/swaths/frequencyA/HH"
!gdalinfo {path}
```

```
Driver: HDF5Image/HDF5 Dataset
Files: ./data/Haywrd_14501_21043_012_210602_L090_CX_129_02.h5
Size is 2640, 15404
Metadata:
  contact=nisarops@jpl.nasa.gov
  Conventions=CF-1.7
  institution=NASA JPL
  mission_name=NISAR
  reference_document=TBD
  title=NISAR L1 SLC Product
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,15404.0)
Upper Right ( 2640.0,    0.0)
Lower Right ( 2640.0,15404.0)
Center      ( 1320.0, 7702.0)
Band 1 Block=128x128 Type=CFloat32, ColorInterp=Undefined
  Metadata:
    science_LSAR_SLC_swaths_frequencyA_HH_description=Focused SLC image (HH)
    science_LSAR_SLC_swaths_frequencyA_HH_units=DN
```

Translate the HH polarization into SLC files. SLC files are Single Look Complex files. More information can be found in the product description here.

[12]:
```
path = f"HDF5:{nisar_list[0]}://science/LSAR/SLC/swaths/frequencyA/HH HH.slc"
!gdal_translate -of ENVI {path}
```

```
Input file size is 2640, 15404
0...10...20...30...40...50...60...70...80...90...100 - done.
```

Check the metadata again.

[13]:
```
!gdalinfo HH.slc
```

```
Driver: ENVI/ENVI .hdr Labelled
Files: HH.slc
       HH.slc.aux.xml
       HH.hdr
Size is 2640, 15404
Metadata:
  contact=nisarops@jpl.nasa.gov
  Conventions=CF-1.7
  institution=NASA JPL
  mission_name=NISAR
  reference_document=TBD
  title=NISAR L1 SLC Product
```

(continues on next page)

(continued from previous page)

```
Image Structure Metadata:
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (    0.0,    0.0)
Lower Left  (    0.0,15404.0)
Upper Right ( 2640.0,    0.0)
Lower Right ( 2640.0,15404.0)
Center      ( 1320.0, 7702.0)
Band 1 Block=2640x1 Type=CFloat32, ColorInterp=Undefined
  Metadata:
    science_LSAR_SLC_swaths_frequencyA_HH_description=Focused SLC image (HH)
    science_LSAR_SLC_swaths_frequencyA_HH_units=DN
```

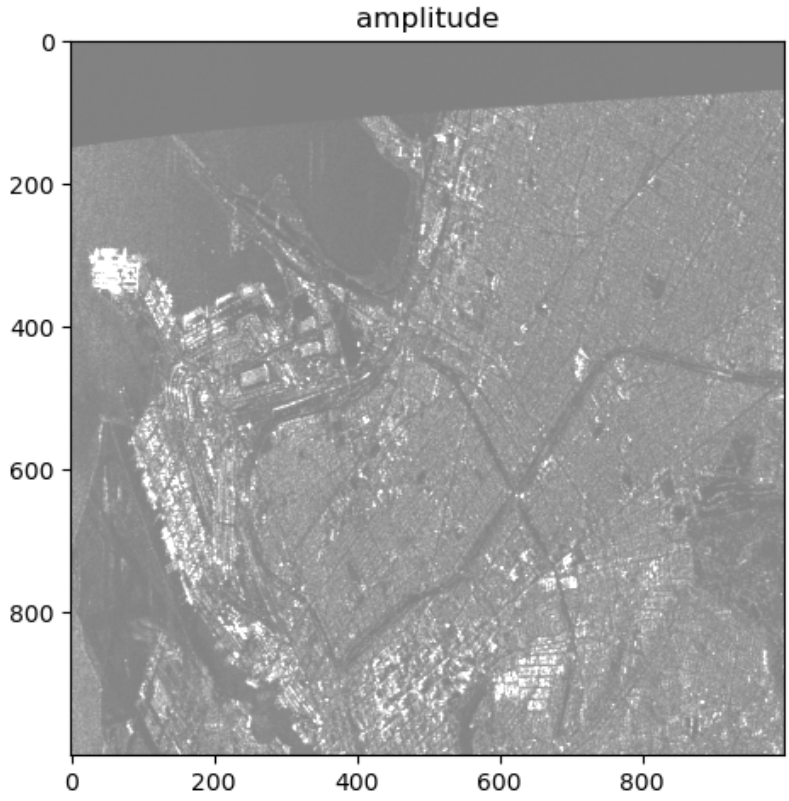
2.8.8 Visualize

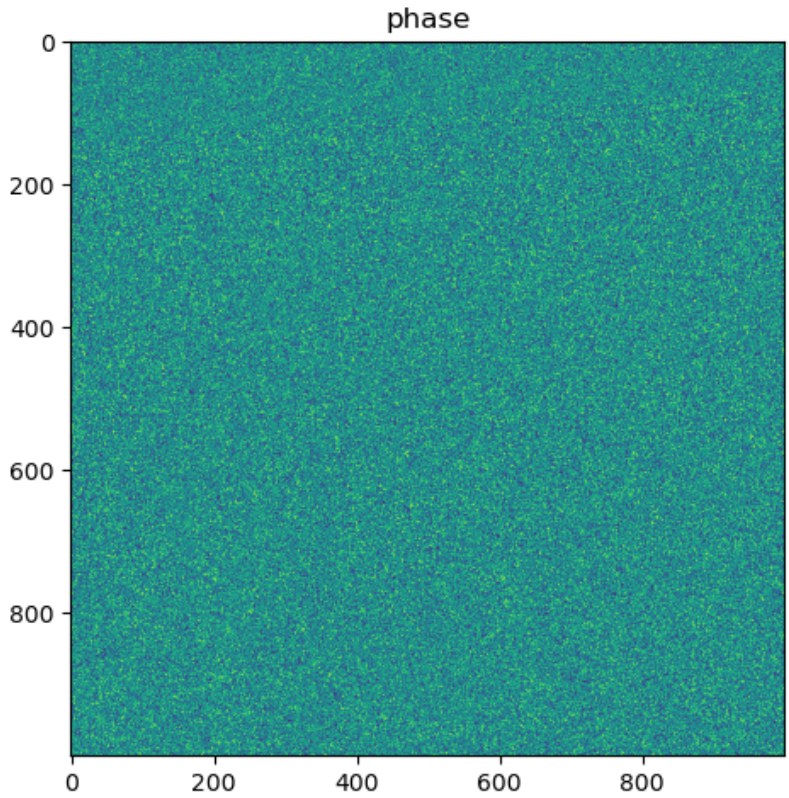
Finally, lets plot amplitude and phase.

[14]:
```
import numpy as np
import rasterio
import matplotlib.pyplot as plt


# Extract a subset of the SLC to display
x0 = 0
y0 = 10
x_offset = 1000
y_offset = 1000

with rasterio.open("HH.slc") as ds:
    # Define the window of data to read
    window = rasterio.windows.Window(x0, y0, x_offset, y_offset)

    # Read the data from the specified window
    slc = ds.read(1, window=window)
    print(slc)

fig = plt.figure(figsize=(14, 12))

# Display amplitude of the slc
ax = fig.add_subplot(2, 1, 1)
ax.imshow(np.abs(slc), vmin=-2, vmax=2, cmap="gray")
ax.set_title("amplitude")

# Display phase of the slc
ax = fig.add_subplot(2, 1, 2)
ax.imshow(np.angle(slc))
ax.set_title("phase")

plt.show()
```

```
/opt/conda/lib/python3.10/site-packages/rasterio/__init__.py:304:␣
↪NotGeoreferencedWarning: Dataset has no geotransform, gcps, or rpcs. The identity␣
↪matrix will be returned.
  dataset = DatasetReader(path, driver=driver, sharing=sharing, **kwargs)
```

```
[[ 2.46178992e-02-0.01162272j  2.06204206e-02+0.0438217j
   2.01302450e-02-0.00866299j ... -3.71712632e-03+0.00754739j
  -3.99701595e-02+0.01584719j  4.39935476e-02+0.03462401j]
 [-5.99635430e-02-0.04907523j -1.78322736e-02-0.04193708j
   4.85753343e-02-0.00152001j ... -3.92718357e-04+0.03611586j
   3.35564204e-02+0.0750533j  -2.58343443e-02+0.00963823j]
 [-1.27040278e-02+0.00876454j  9.38015990e-03+0.01417367j
  -1.09148119e-02-0.00828656j ... -1.72818732e-02-0.08329829j
   5.43021038e-03-0.03845887j -2.69295573e-02-0.03714512j]
 ...
 [-1.18659958e-02-0.04372251j -4.90541667e-01-0.04433485j
  -6.41287342e-02+0.12093218j ... -8.06556106e-01+0.35332313j
   2.51429707e-01-0.24497119j  2.09877625e-01+0.43311965j]
 [ 1.27028033e-01+0.08081798j -1.80030346e-01-0.05467323j
  -1.05361664e+00-0.45684314j ... -8.28622878e-01-0.33322746j
  -1.37368396e-01-0.42675552j  3.07213575e-01-0.02091259j]
 [ 1.09616600e-01-0.02663859j -1.23310655e-01+0.22490016j
  -9.74314928e-01+0.17205611j ... -4.84358639e-01-0.2353034j
  -8.60485807e-02+0.06121469j  2.02866718e-02+0.14383014j]]
```


amplitude
0
200
400
600
800
0
200
400
600
800


phase
0
200
400
600
800
0
200
400
600
800

[ ]:

2.9 AfriSAR Search and Visualize

Authors: Nikita Susan (UAH), Aimee Barciauskas (DevSeed)

Date: January 17, 2023

Description: In this tutorial, we will search for AfriSAR AGB (Above Ground Biomass) data and download a TIFF file from the ORNL DAAC S3. The TIFF file will then be read in with rioxarray and visualized using hvplot.

2.9.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.9.2 About the Data

"This dataset provides gridded estimates of aboveground biomass (AGB) for four sites in Gabon at 0.25 ha (50 m) resolution derived with field measurements and airborne LiDAR data collected from 2010 to 2016. The sites represent a mix of forested, savannah, and some agricultural and disturbed landcover types: Lope site, within Lope National Park; Mabounie, mostly forested site; Mondah Forest, protected area; and the Rabi forest site, part of the Smithsonian Institution of Global Earth Observatories world-wide network of forest plots. Plot-level biophysical measurements of tree diameter and tree height (or estimated by allometry) were performed at 1 ha and 0.25 ha scales on multiple plots at each site and used to derive AGB for each tree and then summed for each plot. Aerial LiDAR scans were used to construct digital elevation models (DEM) and digital surface models (DSM), and then the DEM and DSM were used to construct a canopy height model (CHM) at 1 m resolution. After checking site-plot locations against the CHM, mean canopy height (MCH) was computed over each 0.25 ha. A single regression model relating MCH and AGB estimates, incorporating local height based on the trunk DBH (HD) relationships, was produced for all sites and combined with the CHM layer to construct biomass maps at 0.25 ha resolution." (Source: AfriSAR AGB User Guide)

2.9.3 Additional Resources

- AfriSAR AGB Dataset Landing Page
- Earthdata Search

2.9.4 Import and Install Packages

First, let's import and install packages. If you don't have the packages below installed already, uncomment the following line.

[ ]:
```
# !pip install rioxarray hvplot
```

[38]:
```
import rioxarray
import rasterio as rio
import hvplot.xarray
from maap.maap import MAAP
import boto3
from rasterio.session import AWSSession
import os

import warnings
warnings.filterwarnings("ignore")
```

2.9.5 Search for AfriSAR AGB Data

Using MAAP's searchCollection function and the collection short name, we'll pull in the AfriSAR_AGB_Maps_1681 collection.

[8]:
```
maap = MAAP(maap_host='api.maap-project.org')
```

[83]:
```
results = maap.searchCollection(cmr_host='cmr.earthdata.nasa.gov', short_name='AfriSAR_
↪AGB_Maps_1681')
results
```

[83]:
```
[{'concept-id': 'C2734261660-ORNL_CLOUD',
  'revision-id': '2',
  'format': 'application/echo10+xml',
  'Collection': {'ShortName': 'AfriSAR_AGB_Maps_1681',
   'VersionId': '1',
   'InsertTime': '2022-11-28T00:00:00Z',
   'LastUpdate': '2023-07-17T18:24:39Z',
   'LongName': 'AfriSAR: Aboveground Biomass for Lope, Mabounie, Mondah, and Rabi Sites,␣
↪Gabon',
   'DataSetId': 'AfriSAR: Aboveground Biomass for Lope, Mabounie, Mondah, and Rabi Sites,
↪ Gabon',
   'Description': 'This dataset provides gridded estimates of aboveground biomass (AGB)␣
↪for four sites in Gabon at 0.25 ha (50 m) resolution derived with field measurements␣
↪and airborne LiDAR data collected from 2010 to 2016. The sites represent a mix of␣
↪forested, savannah, and some agricultural and disturbed landcover types: Lope site,␣
↪within Lope National Park; Mabounie, mostly forested site; Mondah Forest, protected␣
↪area; and the Rabi forest site, part of the Smithsonian Institution of Global Earth␣
↪Observatories world-wide network of forest plots. Plot-level biophysical measurements␣
↪of tree diameter and tree height (or estimated by allometry) were performed at 1 ha␣
↪and 0.25 ha scales on multiple plots at each site and used to derive AGB for each tree␣
↪and then summed for each plot. Aerial LiDAR scans were used to construct digital␣
↪elevation models (DEM) and digital surface models (DSM), and then the DEM and DSM were␣
↪used to construct a canopy height model (CHM) at 1 m resolution. After checking site-
↪plot locations against the CHM, mean canopy height (MCH) was computed over each 0.25␣
↪ha. A single regression model relating MCH and AGB estimates, incorporating local␣
↪height based on the trunk DBH (HD) relationships, was produced for all sites and␣
↪combined with the CHM layer to construct biomass maps at 0.25 ha resolution.',
   'DOI': {'DOI': '10.3334/ORNLDAAC/1681', 'Authority': 'https://doi.org'},
   'Orderable': 'false',
   'Visible': 'true',
```

(continues on next page)

(continued from previous page)

```
  'RevisionDate': '2022-11-28T00:00:00Z',
  'ProcessingLevelId': '3',
  'ProcessingLevelDescription': 'Variables mapped on uniform space-time grid scales␣
↪with completeness and consistency',
  'ArchiveCenter': 'ORNL_DAAC',
  'CitationForExternalPublication': 'Saatchi, S.S., J. Chave, N. Labriere, N. Barbier,␣
↪M. Maxime-Rejou, A. Ferraz, and S. Tao. 2019. AfriSAR: Aboveground Biomass for Lope,␣
↪Mabounie, Mondah, and Rabi Sites, Gabon. ORNL DAAC, Oak Ridge, Tennessee, USA. https://
↪doi.org/10.3334/ORNLDAAC/1681',
  'CollectionState': 'COMPLETE',
  'MaintenanceAndUpdateFrequency': 'As needed',
  'UseConstraints': {'LicenseURL': {'URL': 'https://science.nasa.gov/earth-science/
↪earth-science-data/data-information-policy',
    'Description': 'License URL for data use policy',
    'Type': 'Data Use Policy',
    'MimeType': 'text/html'}},
  'Price': '0',
  'DataFormat': 'GeoTIFF',
  'SpatialKeywords': {'Keyword': 'GABON'},
  'Temporal': {'RangeDateTime': {'BeginningDateTime': '2016-02-01T00:00:00Z',
    'EndingDateTime': '2016-03-31T23:59:59Z'}},
  'Contacts': {'Contact': {'Role': 'ARCHIVER',
    'OrganizationName': 'ORNL_DAAC',
    'OrganizationAddresses': {'Address': {'StreetAddress': 'ORNL DAAC User Services␣
↪Office, P.O. Box 2008, MS 6407, Oak Ridge National Laboratory',
      'City': 'Oak Ridge',
      'StateProvince': 'Tennessee',
      'PostalCode': '37831-6407',
      'Country': 'USA'}},
    'OrganizationPhones': {'Phone': {'Number': '(865) 241-3952',
      'Type': 'Direct Line'}},
    'OrganizationEmails': {'Email': 'uso@daac.ornl.gov'}}},
  'ScienceKeywords': {'ScienceKeyword': [{'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'BIOSPHERE',
     'TermKeyword': 'ECOSYSTEMS',
     'VariableLevel1Keyword': {'Value': 'TERRESTRIAL ECOSYSTEMS',
      'VariableLevel2Keyword': {'Value': 'FORESTS'}}},
    {'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'BIOSPHERE',
     'TermKeyword': 'VEGETATION',
     'VariableLevel1Keyword': {'Value': 'BIOMASS'}},
    {'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'BIOSPHERE',
     'TermKeyword': 'VEGETATION',
     'VariableLevel1Keyword': {'Value': 'CARBON'}}]},
  'Platforms': {'Platform': [{'ShortName': 'FIELD SURVEYS',
     'LongName': 'FIELD SURVEYS',
     'Type': 'Field Sites',
     'Instruments': {'Instrument': {'ShortName': 'STEEL MEASURING TAPE',
       'LongName': 'STEEL MEASURING TAPE'}}},
    {'ShortName': 'Airplane',
     'LongName': 'Airplane',
```

(continues on next page)

(continued from previous page)

```
      'Type': 'Jet',
      'Instruments': {'Instrument': {'ShortName': 'LIDAR',
        'LongName': 'Light Detection and Ranging'}}},
     {'ShortName': 'B-200',
      'LongName': 'Beechcraft King Air B-200',
      'Type': 'Propeller',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor'}}}]},
   'Campaigns': {'Campaign': {'ShortName': 'AfriSAR', 'LongName': 'AfriSAR'}},
   'OnlineAccessURLs': {'OnlineAccessURL': {'URL': 'https://daac.ornl.gov/afrisar/
↪AfriSAR_AGB_Maps/',
     'URLDescription': 'This link allows direct data access via Earthdata login'}},
   'OnlineResources': {'OnlineResource': [{'URL': 'https://daac.ornl.gov/AFRISAR/guides/
↪AfriSAR_AGB_Maps.html',
      'Description': 'ORNL DAAC Data Set Documentation',
      'Type': "USER'S GUIDE"},
     {'URL': 'https://doi.org/10.3334/ORNLDAAC/1681',
      'Description': 'Data set Landing Page DOI URL',
      'Type': 'DATA SET LANDING PAGE'},
     {'URL': 'https://data.ornldaac.earthdata.nasa.gov/public/afrisar/AfriSAR_AGB_Maps/
↪comp/AfriSAR_AGB_Maps.pdf',
      'Description': 'AfriSAR: Aboveground Biomass Maps for Lope, Mabounie, Mondah, and␣
↪Rabi, Gabon, 2016: AfriSAR_AGB_Maps.pdf',
      'Type': 'GENERAL DOCUMENTATION'},
     {'URL': 'https://data.ornldaac.earthdata.nasa.gov/public/afrisar/AfriSAR_AGB_Maps/
↪comp/AfriSAR_AGB_Maps_PlotDetails.csv',
      'Description': 'AfriSAR: Aboveground Biomass for Lope, Mabounie, Mondah, and Rabi␣
↪Sites, Gabon: AfriSAR_AGB_Maps_PlotDetails.csv',
      'Type': 'GENERAL DOCUMENTATION'},
     {'URL': 'https://daac.ornl.gov/AFRISAR/guides/AfriSAR_AGB_Maps_Fig1.png',
      'Description': 'Aboveground biomass (AGB) map at 0.25-ha resolution for the Mondah␣
↪study site. Source: Mabounie_AGB_50m.tif',
      'Type': 'GET RELATED VISUALIZATION',
      'MimeType': 'image/png'}]},
   'Spatial': {'SpatialCoverageType': 'HORIZONTAL',
    'HorizontalSpatialDomain': {'Geometry': {'CoordinateSystem': 'CARTESIAN',
      'BoundingRectangle': {'WestBoundingCoordinate': '9.30',
       'NorthBoundingCoordinate': '0.61',
       'EastBoundingCoordinate': '11.64',
       'SouthBoundingCoordinate': '-1.95'}}},
    'GranuleSpatialRepresentation': 'CARTESIAN'},
   'DirectDistributionInformation': {'Region': 'us-west-2',
    'S3BucketAndObjectPrefixName': 's3://ornl-cumulus-prod-protected/afrisar/AfriSAR_AGB_
↪Maps/',
    'S3CredentialsAPIEndpoint': 'https://data.ornldaac.earthdata.nasa.gov/s3credentials',
    'S3CredentialsAPIDocumentationURL': 'https://data.ornldaac.earthdata.nasa.gov/
↪s3credentialsREADME'}}}]
```

Using the searchGranule function and the concept-id from our collection search, we can also discover granules within the collection. For this tutorial, we'll be visualizing the third granule in the collection, so let's retrieve that one. This granule is of the Rabi forest site.

[84]:
```
granules = maap.searchGranule(cmr_host = 'cmr.earthdata.nasa.gov', concept_id =
↪'C2734261660-ORNL_CLOUD')
granules[3]
```

[84]:
```
{'concept-id': 'G2734344223-ORNL_CLOUD',
 'collection-concept-id': 'C2734261660-ORNL_CLOUD',
 'revision-id': '1',
 'format': 'application/echo10+xml',
 'Granule': {'GranuleUR': 'AfriSAR_AGB_Maps.Rabi_AGB_50m.tif',
  'InsertTime': '2022-11-28T00:00:00Z',
  'LastUpdate': '2023-07-17T18:24:45Z',
  'Collection': {'ShortName': 'AfriSAR_AGB_Maps_1681', 'VersionId': '1'},
  'DataGranule': {'DataGranuleSizeInBytes': '14109',
   'SizeMBDataGranule': '0.014109',
   'Checksum': {'Value':
↪'514dca209ed19076e5bdf2595af86af2a76d7a318ad76cc56480fc4a8bb26fba',
    'Algorithm': 'SHA-256'},
   'DayNightFlag': 'BOTH',
   'ProductionDateTime': '2022-11-28T00:00:00Z'},
  'Temporal': {'RangeDateTime': {'BeginningDateTime': '2016-02-01T00:00:00Z',
    'EndingDateTime': '2016-03-31T23:59:59Z'}},
  'Spatial': {'HorizontalSpatialDomain': {'Geometry': {'BoundingRectangle': {
↪'WestBoundingCoordinate': '9.85914',
      'NorthBoundingCoordinate': '-1.90031',
      'EastBoundingCoordinate': '9.90636',
      'SouthBoundingCoordinate': '-1.94602'}}}},
  'MeasuredParameters': {'MeasuredParameter': [{'ParameterName': 'FORESTS'},
    {'ParameterName': 'CARBON'},
    {'ParameterName': 'BIOMASS'}]},
  'Platforms': {'Platform': [{'ShortName': 'FIELD SURVEYS',
     'Instruments': {'Instrument': {'ShortName': 'STEEL MEASURING TAPE'}}},
    {'ShortName': 'Airplane',
     'Instruments': {'Instrument': {'ShortName': 'LIDAR'}}},
    {'ShortName': 'B-200',
     'Instruments': {'Instrument': {'ShortName': 'LVIS'}}}]},
  'Campaigns': {'Campaign': {'ShortName': 'AfriSAR'}},
  'Price': '0',
  'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://data.ornldaac.earthdata.nasa.
↪gov/protected/afrisar/AfriSAR_AGB_Maps/data/Rabi_AGB_50m.tif',
     'URLDescription': 'Download Rabi_AGB_50m.tif',
     'MimeType': 'image/tiff'},
    {'URL': 'https://data.ornldaac.earthdata.nasa.gov/public/afrisar/AfriSAR_AGB_Maps/
↪data/Rabi_AGB_50m.tif.sha256',
     'URLDescription': 'Download Rabi_AGB_50m.tif.sha256'},
    {'URL': 's3://ornl-cumulus-prod-protected/afrisar/AfriSAR_AGB_Maps/data/Rabi_AGB_50m.
↪tif',
     'URLDescription': 'This link provides direct download access via S3 to the granule',
     'MimeType': 'image/tiff'}]},
  'OnlineResources': {'OnlineResource': [{'URL': 'https://daac.ornl.gov/AFRISAR/guides/
↪AfriSAR_AGB_Maps.html',
     'Description': 'ORNL DAAC Data Set Documentation',
     'Type': "USER'S GUIDE"},
    {'URL': 'https://doi.org/10.3334/ORNLDAAC/1681',
```

(continues on next page)

(continued from previous page)

```
     'Description': 'Data set Landing Page DOI URL',
     'Type': 'DATA SET LANDING PAGE'},
    {'URL': 'https://daac.ornl.gov/daacdata/afrisar/AfriSAR_AGB_Maps/comp/AfriSAR_AGB_
↪Maps.pdf',
     'Description': 'Data Set Documentation',
     'Type': 'GENERAL DOCUMENTATION'},
    {'URL': 'https://daac.ornl.gov/daacdata/afrisar/AfriSAR_AGB_Maps/comp/AfriSAR_AGB_
↪Maps_PlotDetails.csv',
     'Description': 'Data Set Documentation',
     'Type': 'GENERAL DOCUMENTATION'},
    {'URL': 'https://data.ornldaac.earthdata.nasa.gov/s3credentials',
     'Description': 'api endpoint to retrieve temporary credentials valid for same-
↪region direct s3 access',
     'Type': 'VIEW RELATED INFORMATION'}]},
  'Orderable': 'false',
  'DataFormat': 'COG'}}
```

2.9.6 Download the Granule File

We'll download our file directly from the ORNL DAAC S3.

Let's pull in the collection and file name for the granule of interest.

[70]:
```
granule_ur=granules[3]['Granule']['GranuleUR'].split(".")
collection_name=granule_ur[0]
file_name=granule_ur[1]
```

[71]:
```
print(f"collection name: {collection_name} | file_name: {file_name}")
```

```
collection name: AfriSAR_AGB_Maps | file_name: Rabi_AGB_50m
```

Next, we will request temporary s3 credentials for the ORNL DAAC. Once these credentials are given, the file can be downloaded to our workspace.

[72]:
```
def get_s3_creds(url):
    return maap.aws.earthdata_s3_credentials(url)

def get_s3_client(s3_cred_endpoint):
    creds=get_s3_creds(s3_cred_endpoint)
    boto3_session = boto3.Session(
            aws_access_key_id=creds['accessKeyId'],
            aws_secret_access_key=creds['secretAccessKey'],
            aws_session_token=creds['sessionToken']
    )
    return boto3_session.client("s3")

def download_s3_file(s3, bucket, collection_name, file_name):
    os.makedirs("/projects/afrisar", exist_ok=True) # create directories, as necessary
    download_path=f"/projects/afrisar/{file_name}.tif"
    s3.download_file(bucket, f"afrisar/{collection_name}/data/{file_name}.tif", download_
↪path)
    return download_path
```

[73]:
```
s3_cred_endpoint= 'https://data.ornldaac.earthdata.nasa.gov/s3credentials'
s3=get_s3_client(s3_cred_endpoint)
```

[74]:
```
bucket="ornl-cumulus-prod-protected"
download_path=download_s3_file(s3, bucket, collection_name, file_name)
download_path
```

[74]:
```
'/projects/afrisar/Rabi_AGB_50m.tif'
```

2.9.7 Read and Visualize

Read in our file with rioxarray…

[75]:
```
da = rioxarray.open_rasterio(download_path)
da = da.squeeze('band', drop=True)
da
```

[75]:
```
<xarray.DataArray (y: 101, x: 105)>
[10605 values with dtype=float32]
Coordinates:
  * x            (x) float64 5.956e+05 5.956e+05 ... 6.007e+05 6.008e+05
  * y            (y) float64 9.79e+06 9.79e+06 9.79e+06 ... 9.785e+06 9.785e+06
    spatial_ref  int64 0
Attributes:
    AREA_OR_POINT:  Area
    _FillValue:     -9999.0
    scale_factor:   1.0
    add_offset:     0.0
```

… and visualize our data using hvplot.

[76]:
```
ds_masked = da.where(da != da._FillValue)

ds_masked.hvplot(
    'x', 'y',
    cmap='viridis',
    frame_height=400,
    frame_width=400
).redim.range(value=(0,da.max().values))
```

[76]:
```
:Image   [x,y]   (value)
```

2.10 LVIS Access and Explore

Authors: Rajat Shinde (UAH), Sheyenne Kirkland (UAH), Alex Mandel (DevSeed), Jamison French (DevSeed), Emile Tenezakis (DevSeed), Brian Freitag (NASA MSFC)

Date: August 22, 2023

Description: In this tutorial, we will explore the LVIS (Land Vegetation and Ice Sensor) based data products and work on accessing these data products using MAAP.

2.10.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: It is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.10.2 Importing Packages

We import the `os` module, import the `MAAP` package, and create a new MAAP class instance.

[ ]:
```
# import os module
import os

# import the MAAP package to handle queries
from maap.maap import MAAP

# import printing package to help display outputs
from pprint import pprint

# invoke the MAAP search client
maap = MAAP()
```

2.10.3 Creating a Data Directory for this Tutorial

[14]:
```
# set data directory path
dataDir = './data'

# check if directory exists -> if directory doesn't exist, directory is created
if not os.path.exists(dataDir):
    os.mkdir(dataDir)
```

After executing the previous cell, we can observe that the data directory has been created and all the files in this tutorial will be downloaded to this directory.

Accessing the LVIS Facility L1B Geolocated Return Energy Waveforms V001

2.10.4 About the Dataset

The LVIS Facility L1B Geolocated Return Energy Waveforms V001 dataset contains Level-1B geolocated return energy waveforms collected by the NASA Land, Vegetation and Ice Sensor (LVIS) Facility, an imaging lidar and camera sensor suite. The short name for this collection is `LVISF1B`. This short name is used for searching this collection in the NASA CMR using the `searchCollection()` method.

2.10.5 Searching the Collection

[15]:
```
lvisf1b_collections = maap.searchCollection(
    short_name='LVISF1B',
    version='1',
    cmr_host='cmr.earthdata.nasa.gov'
)
lvisf1b_collections
```

[15]:
```
[{'concept-id': 'C1723866745-NSIDC_ECS',
  'revision-id': '38',
  'format': 'application/echo10+xml',
  'Collection': {'ShortName': 'LVISF1B',
   'VersionId': '1',
   'InsertTime': '2023-08-22T18:01:08.485Z',
   'LastUpdate': '2023-08-22T18:01:08.485Z',
   'LongName': 'Not provided',
   'DataSetId': 'LVIS Facility L1B Geolocated Return Energy Waveforms V001',
   'Description': 'This data set contains Level-1B geolocated return energy waveforms␣
↪collected by the NASA Land, Vegetation, and Ice Sensor (LVIS) Facility, an imaging␣
↪lidar and camera sensor suite.',
   'DOI': {'DOI': '10.5067/XQJ8PN8FTIDG'},
   'StandardProduct': 'false',
   'RevisionDate': '2023-05-31T00:00:00.000Z',
   'SuggestedUsage': 'Scientific Research',
   'ProcessingCenter': 'NASA/GSFC/SED/ESD/LRSL',
   'ProcessingLevelId': 'Level 1B',
   'ProcessingLevelDescription': 'Sensor units',
   'ArchiveCenter': 'NASA NSIDC DAAC',
   'CollectionState': 'PLANNED',
   'RestrictionComment': ' These data are freely, openly, and fully accessible, provided␣
↪that you are logged into your NASA Earthdata profile (https://urs.earthdata.nasa.gov/).
↪',
   'UseConstraints': {'LicenseText': ' These data are freely, openly, and fully␣
↪available to use without restrictions, provided that you cite the data according to␣
↪the recommended citation at https://nsidc.org/about/use_copyright.html. For more␣
↪information on the NASA EOSDIS Data Use Policy, see https://earthdata.nasa.gov/earth-
↪observation-data/data-use-policy.'},
   'DataFormat': 'HDF5',
   'SpatialKeywords': {'Keyword': ['CANADA',
     'GREENLAND',
     'UNITED STATES OF AMERICA',
     'COSTA RICA',
     'FRENCH GUIANA']},
   'Temporal': {'EndsAtPresentFlag': 'false',
    'RangeDateTime': {'BeginningDateTime': '2018-11-07T00:00:00.000Z'}},
   'Contacts': {'Contact': [{'Role': 'ARCHIVER',
      'OrganizationName': 'NASA NSIDC DAAC',
      'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
        'City': 'Boulder',
        'StateProvince': 'CO',
        'PostalCode': '80309-0449',
```

(continues on next page)

(continued from previous page)

```
        'Country': 'USA'}},
      'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
        'Type': 'Telephone'}},
      'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
     {'Role': 'DISTRIBUTOR',
      'OrganizationName': 'NASA NSIDC DAAC',
      'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
        'City': 'Boulder',
        'StateProvince': 'CO',
        'PostalCode': '80309-0449',
        'Country': 'USA'}},
      'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
        'Type': 'Telephone'}},
      'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
     {'Role': 'PROCESSOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
     {'Role': 'ORIGINATOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
     {'Role': 'TECHNICAL CONTACT',
      'ContactPersons': {'ContactPerson': [{'FirstName': 'NSIDC',
         'MiddleName': 'User',
         'LastName': 'Services',
         'JobPosition': 'TECHNICAL CONTACT'},
        {'FirstName': 'J.',
         'MiddleName': 'Bryan',
         'LastName': 'Blair',
         'JobPosition': 'TECHNICAL CONTACT'},
        {'FirstName': 'Michelle',
         'LastName': 'Hofton',
         'JobPosition': 'TECHNICAL CONTACT'}]}}]},
   'ScienceKeywords': {'ScienceKeyword': {'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'SPECTRAL/ENGINEERING',
     'TermKeyword': 'INFRARED WAVELENGTHS',
     'VariableLevel1Keyword': {'Value': 'SENSOR COUNTS'}}},
   'Platforms': {'Platform': [{'ShortName': 'B-200',
      'LongName': 'Beechcraft King Air B-200',
      'Type': 'Propeller',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
     {'ShortName': 'G-III',
      'LongName': 'Gulfstream III',
      'Type': 'Jet',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
     {'ShortName': 'G-V',
```

(continues on next page)

(continued from previous page)

```
      'LongName': 'Gulfstream V',
      'Type': 'Jet',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
     {'ShortName': 'P-3B',
      'LongName': 'Lockheed P-3B Orion',
      'Type': 'Propeller',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}}]},
   'AdditionalAttributes': {'AdditionalAttribute': [{'Name': 'AircraftID',
      'DataType': 'STRING',
      'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft'},
     {'Name': 'SIPSMetGenVersion',
      'DataType': 'STRING',
      'Description': 'The version of the SIPSMetGen software used to produce the␣
↪metadata file for this granule'},
     {'Name': 'ThemeID',
      'DataType': 'STRING',
      'Description': 'The identifier of the theme under which data are logically grouped
↪'},
     {'Name': 'identifier_product_doi',
      'DataType': 'STRING',
      'Description': 'Digital object identifier that uniquely identifies this data␣
↪product'},
     {'Name': 'identifier_product_doi_authority',
      'DataType': 'STRING',
      'Description': 'URL of the digital object identifier resolving authority'}]},
   'Campaigns': {'Campaign': [{'ShortName': 'ABoVE',
      'LongName': 'Arctic-Boreal Vulnerability Experiment (ABoVE) NASA field campaign',
      'StartDate': '2017-06-29T00:00:00.000Z',
      'EndDate': '2017-07-17T00:00:00.000Z'},
     {'ShortName': 'GEDI',
      'LongName': "NASA's Global Ecosystem Dynamics Investigation",
      'StartDate': '2019-01-01T00:00:00.000Z',
      'EndDate': '2019-12-31T00:00:00.000Z'},
     {'ShortName': 'MULTI_NASA',
      'LongName': 'Operation IceBridge Multiple Campaigns',
      'StartDate': '1993-01-01T00:00:00.000Z',
      'EndDate': '2025-12-31T00:00:00.000Z'}]},
   'SpatialInfo': {'SpatialCoverageType': 'HORIZONTAL'},
   'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://n5eil01u.ecs.nsidc.org/
↪ICEBRIDGE/LVISF1B.001/',
      'URLDescription': 'Direct download via HTTPS protocol.'},
```

(continues on next page)

(continued from previous page)

```
    {'URL': 'https://search.earthdata.nasa.gov/search?q=LVISF1B+V001',
     'URLDescription': "NASA's newest search and order tool for subsetting,␣
↪reprojecting, and reformatting data."},
    {'URL': 'https://nsidc.org/data/data-access-tool/LVISF1B/versions/1/',
     'URLDescription': 'Search and filter data files using a map-based interface'}]},
  'OnlineResources': {'OnlineResource': [{'URL': 'https://doi.org/10.5067/XQJ8PN8FTIDG',
     'Description': 'Provides access to data, documentation, tools, citation␣
↪information, support, and other resources.',
     'Type': 'CollectionURL : DATA SET LANDING PAGE'},
    {'URL': 'https://doi.org/10.5067/XQJ8PN8FTIDG',
     'Description': "Includes a user's guide, supplemental documents like ATBDs and␣
↪academic papers, How Tos, FAQs, etc.",
     'Type': 'VIEW RELATED INFORMATION : GENERAL DOCUMENTATION'}]},
  'Spatial': {'SpatialCoverageType': 'HORIZONTAL',
   'HorizontalSpatialDomain': {'Geometry': {'CoordinateSystem': 'CARTESIAN',
     'BoundingRectangle': {'WestBoundingCoordinate': '-167.0',
      'NorthBoundingCoordinate': '88.0',
      'EastBoundingCoordinate': '18.0',
      'SouthBoundingCoordinate': '2.0'}}},
   'GranuleSpatialRepresentation': 'GEODETIC'}}}]
```

The above cell searches and generates the metadata associated with the LVISF1B collection. The `concept-id` in the above output is important and defines the collection id. Using this collection id (See `COLLECTION_ID` below), we will be searching for the granules in this collection.

2.10.6 Searching and Downloading a Granule

We use `searchGranule()` method for searching granules in a particular collection. Once the list of granules is retrieved, we can download a granule to above-defined path by using the `getData()` method.

[16]:
```
COLLECTION_ID = lvisf1b_collections[0]["concept-id"]

results = maap.searchGranule(
    concept_id=COLLECTION_ID,
    cmr_host="cmr.earthdata.nasa.gov"
)
pprint(f'Got {len(results)} results')


#Validating download
filename = results[0].getData(dataDir)
print(filename)
```

```
'Got 20 results'
./data/LVISF1B_US2018_1107_R2011_067463.h5
```

2.10.7 Validating the Downloaded Product

The downloaded product is stored in a Hierarchical Data Format, Version 5 or HDF5 file format (`.h5`). Such files can be accessed in Python by using the h5py library.

[17]:
```
#Testing the HDF5 files
import h5py
with h5py.File(filename, "r") as f:
    print("Keys: %s" % f.keys())
    # get first object name/key; may or may NOT be a group
    a_group_key = list(f.keys())[0]

    # get the object type for a_group_key: usually group or dataset
    print(type(f[a_group_key]))

    # If a_group_key is a group name,
    # this gets the object names in the group and returns as a list
    data = list(f[a_group_key])

    # If a_group_key is a dataset name,
    # this gets the dataset values and returns as a list
    data = list(f[a_group_key])
    # preferred methods to get dataset values:
    ds_obj = f[a_group_key]      # returns as a h5py dataset object
    ds_arr = f[a_group_key][()]  # returns as a numpy array

    print(ds_arr)
```

```
Keys: <KeysViewHDF5 ['AZIMUTH', 'INCIDENTANGLE', 'LAT0', 'LAT1215', 'LFID', 'LON0',
↪'LON1215', 'RANGE', 'RXWAVE', 'SHOTNUMBER', 'SIGMEAN', 'TIME', 'TXWAVE', 'Z0', 'Z1215',
↪ 'ancillary_data']>
<class 'h5py._hl.dataset.Dataset'>
[159.14177  159.4966   158.7841   ... 127.461716 124.07847  124.752396]
```

Since we are able to access data values from the downloaded granule, it is justified that the downloaded file is a valid data product.

Acessing the ABoVE LVIS L1B Geolocated Return Energy Waveforms V001

2.10.8 About the Dataset

The ABoVE LVIS L1B Geolocated Return Energy Waveforms V001 dataset contains return energy waveform data over Alaska and Western Canada measured by the NASA Land, Vegetation, and Ice Sensor (LVIS), an airborne lidar scanning laser altimeter. The data were collected as part of NASA's Terrestrial Ecology Program campaign, the Arctic-Boreal Vulnerability Experiment (ABoVE). The short name for this collection is `ABLVIS1B`.

2.10.9 Searching the Collection

[18]:
```
ablvis1b_collections = maap.searchCollection(
    short_name='ABLVIS1B',
    version='1',
    cmr_host='cmr.earthdata.nasa.gov'
)
ablvis1b_collections
```

[18]:
```
[{'concept-id': 'C1513105920-NSIDC_ECS',
  'revision-id': '49',
  'format': 'application/echo10+xml',
  'Collection': {'ShortName': 'ABLVIS1B',
   'VersionId': '1',
   'InsertTime': '2023-08-22T16:58:08.001Z',
   'LastUpdate': '2023-08-22T16:58:08.001Z',
   'LongName': 'Not provided',
   'DataSetId': 'ABoVE LVIS L1B Geolocated Return Energy Waveforms V001',
   'Description': "This data set contains return energy waveform data over Alaska and␣
↪Western Canada measured by the NASA Land, Vegetation, and Ice Sensor (LVIS), an␣
↪airborne lidar scanning laser altimeter. The data were collected as part of NASA's␣
↪Terrestrial Ecology Program campaign, the Arctic-Boreal Vulnerability Experiment␣
↪(ABoVE).",
   'DOI': {'DOI': '10.5067/UMRAWS57QAFU'},
   'StandardProduct': 'false',
   'RevisionDate': '2023-07-05T00:00:00.000Z',
   'SuggestedUsage': 'Scientific Research',
   'ProcessingCenter': 'NASA/GSFC/SED/ESD/LRSL',
   'ProcessingLevelId': 'Level 1B',
   'ProcessingLevelDescription': 'Sensor units',
   'ArchiveCenter': 'NASA NSIDC DAAC',
   'CollectionState': 'PLANNED',
   'RestrictionComment': ' These data are freely, openly, and fully accessible, provided␣
↪that you are logged into your NASA Earthdata profile (https://urs.earthdata.nasa.gov/).
↪',
   'UseConstraints': {'LicenseText': ' These data are freely, openly, and fully␣
↪available to use without restrictions, provided that you cite the data according to␣
↪the recommended citation at https://nsidc.org/about/use_copyright.html. For more␣
↪information on the NASA EOSDIS Data Use Policy, see https://earthdata.nasa.gov/earth-
↪observation-data/data-use-policy.'},
   'DataFormat': 'HDF5',
   'SpatialKeywords': {'Keyword': ['ALASKA', 'CANADA']},
   'Temporal': {'EndsAtPresentFlag': 'false',
    'RangeDateTime': {'BeginningDateTime': '2017-06-29T00:00:00.000Z',
     'EndingDateTime': '2017-07-17T23:59:59.999Z'}},
   'Contacts': {'Contact': [{'Role': 'ARCHIVER',
      'OrganizationName': 'NASA NSIDC DAAC',
      'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
        'City': 'Boulder',
        'StateProvince': 'CO',
        'PostalCode': '80309-0449',
        'Country': 'USA'}},
```

(continues on next page)

(continued from previous page)

```
      'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
        'Type': 'Telephone'}},
      'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
     {'Role': 'DISTRIBUTOR',
      'OrganizationName': 'NASA NSIDC DAAC',
      'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
        'City': 'Boulder',
        'StateProvince': 'CO',
        'PostalCode': '80309-0449',
        'Country': 'USA'}},
      'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
        'Type': 'Telephone'}},
      'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
     {'Role': 'PROCESSOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
     {'Role': 'ORIGINATOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
     {'Role': 'TECHNICAL CONTACT',
      'ContactPersons': {'ContactPerson': [{'FirstName': 'NSIDC',
         'MiddleName': 'User',
         'LastName': 'Services',
         'JobPosition': 'TECHNICAL CONTACT'},
        {'FirstName': 'J.',
         'MiddleName': 'Bryan',
         'LastName': 'Blair',
         'JobPosition': 'TECHNICAL CONTACT'},
        {'FirstName': 'Michelle',
         'LastName': 'Hofton',
         'JobPosition': 'TECHNICAL CONTACT'}]}}]},
   'ScienceKeywords': {'ScienceKeyword': {'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'SPECTRAL/ENGINEERING',
     'TermKeyword': 'INFRARED WAVELENGTHS',
     'VariableLevel1Keyword': {'Value': 'SENSOR COUNTS'}}},
   'Platforms': {'Platform': [{'ShortName': 'AIRCRAFT',
      'LongName': 'Not provided',
      'Type': 'Aircraft',
      'Instruments': {'Instrument': [{'ShortName': 'ALTIMETERS',
         'Technique': 'instrument',
         'NumberOfSensors': '1',
         'Sensors': {'Sensor': {'ShortName': 'ALTIMETERS'}}},
        {'ShortName': 'LASERS',
         'LongName': 'Light Amplification by Stimulated Emission of Radiation',
         'Technique': 'instrument',
         'NumberOfSensors': '1',
         'Sensors': {'Sensor': {'ShortName': 'LASERS',
           'LongName': 'Light Amplification by Stimulated Emission of Radiation'}}}]}},
     {'ShortName': 'B-200',
      'LongName': 'Beechcraft King Air B-200',
      'Type': 'Propeller',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
```

(continues on next page)

(continued from previous page)

```
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
    {'ShortName': 'C-130',
     'LongName': 'Lockheed C-130 Hercules',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
    {'ShortName': 'DC-8',
     'LongName': 'Douglas DC-8',
     'Type': 'Aircraft',
     'Characteristics': {'Characteristic': {'Name': 'AircraftID',
       'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft',
       'DataType': 'STRING',
       'Unit': 'Not Applicable',
       'Value': 'N817NA'}},
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
    {'ShortName': 'G-V',
     'LongName': 'Gulfstream V',
     'Type': 'Jet',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
    {'ShortName': 'HU-25C',
     'LongName': 'Dassault HU-25C Guardian',
     'Type': 'Jet',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
    {'ShortName': 'P-3B',
     'LongName': 'Lockheed P-3B Orion',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
```

(continues on next page)

(continued from previous page)

```
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
     {'ShortName': 'RQ-4',
      'LongName': 'Northrop Grumman RQ-4 Global Hawk',
      'Type': 'Uncrewed Aerial Vehicles',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}}]},
   'AdditionalAttributes': {'AdditionalAttribute': [{'Name': 'AircraftID',
      'DataType': 'STRING',
      'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft'},
     {'Name': 'SIPSMetGenVersion',
      'DataType': 'STRING',
      'Description': 'The version of the SIPSMetGen software used to produce the␣
↪metadata file for this granule'},
     {'Name': 'ThemeID',
      'DataType': 'STRING',
      'Description': 'The identifier of the theme under which data are logically grouped
↪'},
     {'Name': 'identifier_product_doi',
      'DataType': 'STRING',
      'Description': 'Digital object identifier that uniquely identifies this data␣
↪product'},
     {'Name': 'identifier_product_doi_authority',
      'DataType': 'STRING',
      'Description': 'URL of the digital object identifier resolving authority'}]},
   'Campaigns': {'Campaign': {'ShortName': 'ABoVE',
     'LongName': 'Arctic-Boreal Vulnerability Experiment (ABoVE) NASA field campaign',
     'StartDate': '2017-06-29T00:00:00.000Z',
     'EndDate': '2017-07-17T00:00:00.000Z'}},
   'SpatialInfo': {'SpatialCoverageType': 'HORIZONTAL'},
   'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://n5eil01u.ecs.nsidc.org/
↪ICEBRIDGE/ABLVIS1B.001',
      'URLDescription': 'Direct download via HTTPS protocol.'},
     {'URL': 'https://search.earthdata.nasa.gov/search?q=ABLVIS1B+V001',
      'URLDescription': "NASA's newest search and order tool for subsetting,␣
↪reprojecting, and reformatting data."},
     {'URL': 'https://nsidc.org/icebridge/portal/map',
      'URLDescription': 'Tool to visualize, search, and download IceBridge data.'},
     {'URL': 'https://nsidc.org/data/data-access-tool/ABLVIS1B/versions/1/',
      'URLDescription': 'Search and filter data files using a map-based interface'}]},
   'OnlineResources': {'OnlineResource': [{'URL': 'https://doi.org/10.5067/UMRAWS57QAFU',
      'Description': 'Provides access to data, documentation, tools, citation␣
↪information, support, and other resources.',
      'Type': 'CollectionURL : DATA SET LANDING PAGE'},
     {'URL': 'https://doi.org/10.5067/UMRAWS57QAFU',
      'Description': "Includes a user's guide, supplemental documents like ATBDs and␣
↪academic papers, How Tos, FAQs, etc.",
      'Type': 'VIEW RELATED INFORMATION : GENERAL DOCUMENTATION'}]},
```

(continues on next page)

(continued from previous page)

```
   'Spatial': {'SpatialCoverageType': 'HORIZONTAL',
    'HorizontalSpatialDomain': {'Geometry': {'CoordinateSystem': 'CARTESIAN',
      'BoundingRectangle': {'WestBoundingCoordinate': '-158.0',
       'NorthBoundingCoordinate': '72.0',
       'EastBoundingCoordinate': '-104.0',
       'SouthBoundingCoordinate': '48.0'}}},
    'GranuleSpatialRepresentation': 'GEODETIC'}}}]
```

Based on the output of the above cell, we can retrieve the `concept-id` as collection id and use in the subsequent cells for downloading the granules from this collection.

2.10.10 Searching and Downloading a Granule

[20]:
```
COLLECTION_ID = ablvis1b_collections[0]['concept-id']

results = maap.searchGranule(
    concept_id=COLLECTION_ID,
    cmr_host="cmr.earthdata.nasa.gov"
)
pprint(f'Got {len(results)} results')


#Validating download
filename = results[0].getData(dataDir)
print(filename)
```

```
'Got 20 results'
./data/LVIS1B_ABoVE2017_0629_R1803_056233.h5
```

2.10.11 Validating the Downloaded Product

Similar to the `LVISF1B` dataset (as discussed above), `ABLVIS1B` dataset has products stored as `.h5` files. We will be following the same steps as described above to validate the downloaded product.

[21]:
```
#Testing the HDF5 files
import h5py
with h5py.File(filename, "r") as f:
    print("Keys: %s" % f.keys())
    # get first object name/key; may or may NOT be a group
    a_group_key = list(f.keys())[0]

    # get the object type for a_group_key: usually group or dataset
    print(type(f[a_group_key]))

    # If a_group_key is a group name,
    # this gets the object names in the group and returns as a list
    data = list(f[a_group_key])

    # If a_group_key is a dataset name,
    # this gets the dataset values and returns as a list
```

(continues on next page)

(continued from previous page)

```
    data = list(f[a_group_key])
    # preferred methods to get dataset values:
    ds_obj = f[a_group_key]      # returns as a h5py dataset object
    ds_arr = f[a_group_key][()]  # returns as a numpy array

    print(ds_arr)
```

```
Keys: <KeysViewHDF5 ['AZIMUTH', 'INCIDENTANGLE', 'LAT0', 'LAT1215', 'LFID', 'LON0',
↪'LON1215', 'RANGE', 'RXWAVE', 'SHOTNUMBER', 'SIGMEAN', 'TIME', 'TXWAVE', 'Z0', 'Z1215',
↪ 'ancillary_data']>
<class 'h5py._hl.dataset.Dataset'>
[209.55528 209.98833 209.2643  ... 181.83081 183.49066 182.28035]
```

Accessing LVIS Facility L2 Geolocated Surface Elevation and Canopy Height Product V001

2.10.12 About the Dataset

The LVIS Facility L2 Geolocated Surface Elevation and Canopy Height Product V001 dataset contains Level-2 geolocated surface elevation and canopy height measurements collected by the NASA Land, Vegetation, and Ice Sensor (LVIS) Facility, an imaging lidar and camera sensor suite. The short name for this collection is `LVISF2`.

2.10.13 Searching the Collection

[23]:
```
lvisf2_collections = maap.searchCollection(
    short_name='LVISF2',
    version='1',
    cmr_host='cmr.earthdata.nasa.gov'
)
lvisf2_collections
```

[23]:
```
[{'concept-id': 'C1723866830-NSIDC_ECS',
  'revision-id': '29',
  'format': 'application/echo10+xml',
  'Collection': {'ShortName': 'LVISF2',
   'VersionId': '1',
   'InsertTime': '2023-08-22T16:21:40.702Z',
   'LastUpdate': '2023-08-22T16:21:40.702Z',
   'LongName': 'Not provided',
   'DataSetId': 'LVIS Facility L2 Geolocated Surface Elevation and Canopy Height Product␣
↪V001',
   'Description': 'This data set contains Level-2 geolocated surface elevation and␣
↪canopy height measurements collected by the NASA Land, Vegetation, and Ice Sensor␣
↪(LVIS) Facility, an imaging lidar and camera sensor suite.',
   'DOI': {'DOI': '10.5067/VP7J20HJQISD'},
   'StandardProduct': 'false',
   'RevisionDate': '2023-05-31T00:00:00.000Z',
   'SuggestedUsage': 'Scientific Research',
   'ProcessingCenter': 'NASA/GSFC/SED/ESD/LRSL',
   'ProcessingLevelId': 'Level 2',
   'ProcessingLevelDescription': 'Derived geophysical variables',
```

(continues on next page)

(continued from previous page)

```
   'ArchiveCenter': 'NASA NSIDC DAAC',
   'CollectionState': 'PLANNED',
   'RestrictionComment': ' These data are freely, openly, and fully accessible, provided␣
↪that you are logged into your NASA Earthdata profile (https://urs.earthdata.nasa.gov/).
↪',
   'UseConstraints': {'LicenseText': ' These data are freely, openly, and fully␣
↪available to use without restrictions, provided that you cite the data according to␣
↪the recommended citation at https://nsidc.org/about/use_copyright.html. For more␣
↪information on the NASA EOSDIS Data Use Policy, see https://earthdata.nasa.gov/earth-
↪observation-data/data-use-policy.'},
   'DataFormat': 'ASCII',
   'SpatialKeywords': {'Keyword': ['ALASKA',
     'CANADA',
     'GREENLAND',
     'UNITED STATES OF AMERICA',
     'COSTA RICA',
     'FRENCH GUIANA']},
   'Temporal': {'EndsAtPresentFlag': 'false',
    'RangeDateTime': {'BeginningDateTime': '2018-11-07T00:00:00.000Z'}},
   'Contacts': {'Contact': [{'Role': 'ARCHIVER',
      'OrganizationName': 'NASA NSIDC DAAC',
      'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
        'City': 'Boulder',
        'StateProvince': 'CO',
        'PostalCode': '80309-0449',
        'Country': 'USA'}},
      'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
        'Type': 'Telephone'}},
      'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
     {'Role': 'DISTRIBUTOR',
      'OrganizationName': 'NASA NSIDC DAAC',
      'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
        'City': 'Boulder',
        'StateProvince': 'CO',
        'PostalCode': '80309-0449',
        'Country': 'USA'}},
      'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
        'Type': 'Telephone'}},
      'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
     {'Role': 'PROCESSOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
     {'Role': 'ORIGINATOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
     {'Role': 'TECHNICAL CONTACT',
      'ContactPersons': {'ContactPerson': [{'FirstName': 'NSIDC',
         'MiddleName': 'User',
         'LastName': 'Services',
         'JobPosition': 'TECHNICAL CONTACT'},
        {'FirstName': 'J.',
         'MiddleName': 'Bryan',
         'LastName': 'Blair',
         'JobPosition': 'TECHNICAL CONTACT'},
```

(continues on next page)

(continued from previous page)

```
       {'FirstName': 'Michelle',
        'LastName': 'Hofton',
        'JobPosition': 'TECHNICAL CONTACT'}]}}]},
  'ScienceKeywords': {'ScienceKeyword': [{'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'CRYOSPHERE',
     'TermKeyword': 'GLACIERS/ICE SHEETS',
     'VariableLevel1Keyword': {'Value': 'GLACIER ELEVATION/ICE SHEET ELEVATION'}},
    {'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'CRYOSPHERE',
     'TermKeyword': 'SEA ICE',
     'VariableLevel1Keyword': {'Value': 'SEA ICE ELEVATION'}},
    {'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'LAND SURFACE',
     'TermKeyword': 'TOPOGRAPHY',
     'VariableLevel1Keyword': {'Value': 'TERRAIN ELEVATION'}},
    {'CategoryKeyword': 'EARTH SCIENCE',
     'TopicKeyword': 'BIOSPHERE',
     'TermKeyword': 'VEGETATION',
     'VariableLevel1Keyword': {'Value': 'CANOPY CHARACTERISTICS',
      'VariableLevel2Keyword': {'Value': 'VEGETATION HEIGHT'}}}]},
  'Platforms': {'Platform': [{'ShortName': 'B-200',
     'LongName': 'Beechcraft King Air B-200',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'G-III',
     'LongName': 'Gulfstream III',
     'Type': 'Jet',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'G-V',
     'LongName': 'Gulfstream V',
     'Type': 'Jet',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'P-3B',
     'LongName': 'Lockheed P-3B Orion',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
```

(continues on next page)

(continued from previous page)

```
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}]},
  'AdditionalAttributes': {'AdditionalAttribute': [{'Name': 'AircraftID',
     'DataType': 'STRING',
     'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft'},
    {'Name': 'SIPSMetGenVersion',
     'DataType': 'STRING',
     'Description': 'The version of the SIPSMetGen software used to produce the␣
↪metadata file for this granule'},
    {'Name': 'ThemeID',
     'DataType': 'STRING',
     'Description': 'The identifier of the theme under which data are logically grouped
↪'},
    {'Name': 'identifier_product_doi',
     'DataType': 'STRING',
     'Description': 'Digital object identifier that uniquely identifies this data␣
↪product'},
    {'Name': 'identifier_product_doi_authority',
     'DataType': 'STRING',
     'Description': 'URL of the digital object identifier resolving authority'}]},
  'Campaigns': {'Campaign': [{'ShortName': 'ABoVE',
     'LongName': 'Arctic-Boreal Vulnerability Experiment (ABoVE) NASA field campaign',
     'StartDate': '2017-06-29T00:00:00.000Z',
     'EndDate': '2017-07-17T00:00:00.000Z'},
    {'ShortName': 'GEDI',
     'LongName': "NASA's Global Ecosystem Dynamics Investigation",
     'StartDate': '2019-01-01T00:00:00.000Z',
     'EndDate': '2019-12-31T00:00:00.000Z'},
    {'ShortName': 'MULTI_NASA',
     'LongName': 'Operation IceBridge Multiple Campaigns',
     'StartDate': '1993-01-01T00:00:00.000Z',
     'EndDate': '2025-12-31T00:00:00.000Z'}]},
  'SpatialInfo': {'SpatialCoverageType': 'HORIZONTAL'},
  'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://n5eil01u.ecs.nsidc.org/
↪ICEBRIDGE/LVISF2.001/',
     'URLDescription': 'Direct download via HTTPS protocol.'},
    {'URL': 'https://search.earthdata.nasa.gov/search?q=LVISF2+V001',
     'URLDescription': "NASA's newest search and order tool for subsetting,␣
↪reprojecting, and reformatting data."},
    {'URL': 'https://nsidc.org/data/data-access-tool/LVISF2/versions/1/',
     'URLDescription': 'Search and filter data files using a map-based interface'}]},
  'OnlineResources': {'OnlineResource': [{'URL': 'https://doi.org/10.5067/VP7J20HJQISD',
     'Description': 'Provides access to data, documentation, tools, citation␣
↪information, support, and other resources.',
     'Type': 'CollectionURL : DATA SET LANDING PAGE'},
    {'URL': 'https://doi.org/10.5067/VP7J20HJQISD',
     'Description': "Includes a user's guide, supplemental documents like ATBDs and␣
↪academic papers, How Tos, FAQs, etc.",
     'Type': 'VIEW RELATED INFORMATION : GENERAL DOCUMENTATION'}]},
```

(continues on next page)

(continued from previous page)

```
   'Spatial': {'SpatialCoverageType': 'HORIZONTAL',
    'HorizontalSpatialDomain': {'Geometry': {'CoordinateSystem': 'CARTESIAN',
      'BoundingRectangle': {'WestBoundingCoordinate': '-167.0',
       'NorthBoundingCoordinate': '88.0',
       'EastBoundingCoordinate': '18.0',
       'SouthBoundingCoordinate': '2.0'}}},
    'GranuleSpatialRepresentation': 'GEODETIC'}}}]
```

2.10.14 Searching and Downloading a Granule

[24]:
```
COLLECTION_ID = lvisf2_collections[0]['concept-id']

results = maap.searchGranule(
    concept_id=COLLECTION_ID,
    cmr_host="cmr.earthdata.nasa.gov"
)
pprint(f'Got {len(results)} results')


#Validating download
filename = results[0].getData(dataDir)
print(filename)
```

```
'Got 20 results'
./data/LVISF2_US2018_1107_R2011_067463.TXT
```

We follow the same steps as above for searching and downloading a granule from the LVISF2 collection. However, as we observe, the downloaded product in this case is a `.txt` file. So, we will be using the numpy Python package for accessing the downloaded text file.

2.10.15 Validating the Downloaded Product

[25]:
```
#Testing the TXT files from the LVIS collections
import numpy as np
data = np.genfromtxt(filename)
print(data)
```

```
[[1.95842916e+09 5.93768900e+06 6.74631929e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95842916e+09 5.93769000e+06 6.74631931e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95842916e+09 5.93769100e+06 6.74631934e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 ...
 [1.95842916e+09 6.49713000e+06 6.76030543e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95842916e+09 6.49713100e+06 6.76030545e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95842916e+09 6.49713200e+06 6.76030548e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]]
```

As we can observe from the output of the above cell, the data values are accessible from the downloaded product. Hence, validating the downloaded file.

Accessing the ABoVE LVIS L2 Geolocated Surface Elevation Product V001

2.10.16 About the Dataset

This dataset contains surface elevation data over Alaska and Western Canada measured by the NASA Land, Vegetation, and Ice Sensor (LVIS), an airborne lidar scanning laser altimeter. The data were collected as part of NASA's Terrestrial Ecology Program campaign, the Arctic-Boreal Vulnerability Experiment (ABoVE). The short name for this collection is `ABLVIS2`.

2.10.17 Searching the Collection

[26]:
```
ablvis2_collections = maap.searchCollection(
    short_name='ABLVIS2',
    version='1',
    cmr_host='cmr.earthdata.nasa.gov'
)
ablvis2_collections
```

[26]:
```
[{'concept-id': 'C1513105984-NSIDC_ECS',
  'revision-id': '43',
  'format': 'application/echo10+xml',
  'Collection': {'ShortName': 'ABLVIS2',
   'VersionId': '1',
   'InsertTime': '2023-08-22T16:58:08.004Z',
   'LastUpdate': '2023-08-22T16:58:08.004Z',
   'LongName': 'Not provided',
   'DataSetId': 'ABoVE LVIS L2 Geolocated Surface Elevation Product V001',
   'Description': "This data set contains surface elevation data over Alaska and Western␣
↪Canada measured by the NASA Land, Vegetation, and Ice Sensor (LVIS), an airborne lidar␣
↪scanning laser altimeter. The data were collected as part of NASA's Terrestrial␣
↪Ecology Program campaign, the Arctic-Boreal Vulnerability Experiment (ABoVE).",
   'DOI': {'DOI': '10.5067/IA5WAX7K3YGY'},
   'StandardProduct': 'false',
   'RevisionDate': '2023-05-31T00:00:00.000Z',
   'SuggestedUsage': 'Scientific Research',
   'ProcessingCenter': 'NASA/GSFC/SED/ESD/LRSL',
   'ProcessingLevelId': 'Level 2',
   'ProcessingLevelDescription': 'Derived geophysical variables',
   'ArchiveCenter': 'NASA NSIDC DAAC',
   'CollectionState': 'PLANNED',
   'RestrictionComment': ' These data are freely, openly, and fully accessible, provided␣
↪that you are logged into your NASA Earthdata profile (https://urs.earthdata.nasa.gov/).
↪',
   'UseConstraints': {'LicenseText': ' These data are freely, openly, and fully␣
↪available to use without restrictions, provided that you cite the data according to␣
↪the recommended citation at https://nsidc.org/about/use_copyright.html. For more␣
↪information on the NASA EOSDIS Data Use Policy, see https://earthdata.nasa.gov/earth-
↪observation-data/data-use-policy.'},
   'DataFormat': 'ASCII',
```

(continues on next page)

(continued from previous page)

```
  'SpatialKeywords': {'Keyword': ['CANADA', 'ALASKA']},
  'Temporal': {'EndsAtPresentFlag': 'false',
   'RangeDateTime': {'BeginningDateTime': '2017-06-29T00:00:00.000Z',
    'EndingDateTime': '2017-07-17T23:59:59.999Z'}},
  'Contacts': {'Contact': [{'Role': 'ARCHIVER',
     'OrganizationName': 'NASA NSIDC DAAC',
     'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
       'City': 'Boulder',
       'StateProvince': 'CO',
       'PostalCode': '80309-0449',
       'Country': 'USA'}},
     'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
       'Type': 'Telephone'}},
     'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
    {'Role': 'DISTRIBUTOR',
     'OrganizationName': 'NASA NSIDC DAAC',
     'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
       'City': 'Boulder',
       'StateProvince': 'CO',
       'PostalCode': '80309-0449',
       'Country': 'USA'}},
     'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
       'Type': 'Telephone'}},
     'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
    {'Role': 'PROCESSOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
    {'Role': 'ORIGINATOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
    {'Role': 'TECHNICAL CONTACT',
     'ContactPersons': {'ContactPerson': [{'FirstName': 'J.',
        'MiddleName': 'Bryan',
        'LastName': 'Blair',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'Michelle',
        'LastName': 'Hofton',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'NSIDC',
        'MiddleName': 'User',
        'LastName': 'Services',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'J.',
        'MiddleName': 'Bryan',
        'LastName': 'Blair',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'Michelle',
        'LastName': 'Hofton',
        'JobPosition': 'TECHNICAL CONTACT'}]}}]},
  'ScienceKeywords': {'ScienceKeyword': {'CategoryKeyword': 'EARTH SCIENCE',
    'TopicKeyword': 'LAND SURFACE',
    'TermKeyword': 'TOPOGRAPHY',
    'VariableLevel1Keyword': {'Value': 'TERRAIN ELEVATION'}}},
  'Platforms': {'Platform': [{'ShortName': 'AIRCRAFT',
```

(continues on next page)

(continued from previous page)

```
     'LongName': 'Not provided',
     'Type': 'Aircraft',
     'Instruments': {'Instrument': [{'ShortName': 'ALTIMETERS',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'ALTIMETERS'}}},
       {'ShortName': 'LASERS',
        'LongName': 'Light Amplification by Stimulated Emission of Radiation',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LASERS',
          'LongName': 'Light Amplification by Stimulated Emission of Radiation'}}}]}},
    {'ShortName': 'B-200',
     'LongName': 'Beechcraft King Air B-200',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'C-130',
     'LongName': 'Lockheed C-130 Hercules',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'DC-8',
     'LongName': 'Douglas DC-8',
     'Type': 'Aircraft',
     'Characteristics': {'Characteristic': {'Name': 'AircraftID',
       'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft',
       'DataType': 'STRING',
       'Unit': 'Not Applicable',
       'Value': 'N817NA'}},
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'G-V',
     'LongName': 'Gulfstream V',
     'Type': 'Jet',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
```

(continues on next page)

(continued from previous page)

```
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
     {'ShortName': 'HU-25C',
      'LongName': 'Dassault HU-25C Guardian',
      'Type': 'Jet',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
     {'ShortName': 'P-3B',
      'LongName': 'Lockheed P-3B Orion',
      'Type': 'Propeller',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
     {'ShortName': 'RQ-4',
      'LongName': 'Northrop Grumman RQ-4 Global Hawk',
      'Type': 'Uncrewed Aerial Vehicles',
      'Instruments': {'Instrument': {'ShortName': 'LVIS',
        'LongName': 'Land, Vegetation, and Ice Sensor',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}]},
   'AdditionalAttributes': {'AdditionalAttribute': [{'Name': 'AircraftID',
      'DataType': 'STRING',
      'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft'},
     {'Name': 'SIPSMetGenVersion',
      'DataType': 'STRING',
      'Description': 'The version of the SIPSMetGen software used to produce the␣
↪metadata file for this granule'},
     {'Name': 'ThemeID',
      'DataType': 'STRING',
      'Description': 'The identifier of the theme under which data are logically grouped
↪'},
     {'Name': 'identifier_product_doi',
      'DataType': 'STRING',
      'Description': 'Digital object identifier that uniquely identifies this data␣
↪product'},
     {'Name': 'identifier_product_doi_authority',
      'DataType': 'STRING',
      'Description': 'URL of the digital object identifier resolving authority'}]},
   'Campaigns': {'Campaign': {'ShortName': 'ABoVE',
     'LongName': 'Arctic-Boreal Vulnerability Experiment (ABoVE) NASA field campaign',
     'StartDate': '2017-06-29T00:00:00.000Z',
     'EndDate': '2017-07-17T00:00:00.000Z'}},
```

(continues on next page)

(continued from previous page)

```
   'SpatialInfo': {'SpatialCoverageType': 'HORIZONTAL'},
   'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://n5eil01u.ecs.nsidc.org/
↪ICEBRIDGE/ABLVIS2.001/',
      'URLDescription': 'Direct download via HTTPS protocol.'},
     {'URL': 'https://search.earthdata.nasa.gov/search?q=ABLVIS2+V001',
      'URLDescription': "NASA's newest search and order tool for subsetting,␣
↪reprojecting, and reformatting data."},
     {'URL': 'https://nsidc.org/icebridge/portal/map',
      'URLDescription': 'Tool to visualize, search, and download IceBridge data.'},
     {'URL': 'https://nsidc.org/data/data-access-tool/ABLVIS2/versions/1/',
      'URLDescription': 'Search and filter data files using a map-based interface'}]},
   'OnlineResources': {'OnlineResource': [{'URL': 'https://doi.org/10.5067/IA5WAX7K3YGY',
      'Description': 'Provides access to data, documentation, tools, citation␣
↪information, support, and other resources.',
      'Type': 'CollectionURL : DATA SET LANDING PAGE'},
     {'URL': 'https://doi.org/10.5067/IA5WAX7K3YGY',
      'Description': "Includes a user's guide, supplemental documents like ATBDs and␣
↪academic papers, How Tos, FAQs, etc.",
      'Type': 'VIEW RELATED INFORMATION : GENERAL DOCUMENTATION'}]},
   'Spatial': {'SpatialCoverageType': 'HORIZONTAL',
    'HorizontalSpatialDomain': {'Geometry': {'CoordinateSystem': 'CARTESIAN',
      'BoundingRectangle': {'WestBoundingCoordinate': '-158.0',
       'NorthBoundingCoordinate': '72.0',
       'EastBoundingCoordinate': '-104.0',
       'SouthBoundingCoordinate': '48.0'}}},
    'GranuleSpatialRepresentation': 'GEODETIC'}}}]
```

2.10.18 Searching and Downloading a Granule

We follow the same steps as above.

[27]:
```
COLLECTION_ID = ablvis2_collections[0]['concept-id']

results = maap.searchGranule(
    concept_id=COLLECTION_ID,
    cmr_host="cmr.earthdata.nasa.gov"
)
pprint(f'Got {len(results)} results')


#Validating download
filename = results[0].getData(dataDir)
print(filename)
```

```
'Got 20 results'
./data/LVIS2_ABoVE2017_0629_R1803_056233.TXT
```

2.10.19 Validating the Dataset

Since the output for this data product is also downloaded as a text file, we will follow the same steps as decribed above for the `LVISF2` data product.

[28]:
```
#Testing the downloaded TXT file
import numpy as np
data = np.genfromtxt(filename)
print(data)
```

```
[[1.95793304e+09 3.99310900e+06 5.62334890e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95793304e+09 3.99311000e+06 5.62334890e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95793304e+09 3.99311100e+06 5.62334890e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 ...
 [1.95793306e+09 5.43227700e+06 5.65932850e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95793306e+09 5.43227900e+06 5.65932850e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]
 [1.95793306e+09 5.43228200e+06 5.65932860e+04 ... 1.00000000e+00
  1.00000000e+00 1.00000000e+00]]
```

As observed from the output of the above cell, the data values are accessible from the downloaded product. Hence, validating the downloaded file.

Accessing the AfriSAR LVIS L1B Geolocated Return Energy Waveforms V001

2.10.20 About the Dataset

The AfriSAR LVIS L1B Geolocated Return Energy Waveforms V001 dataset contains return energy waveform data over Gabon, Africa. The measurements were taken by the NASA Land, Vegetation, and Ice Sensor (LVIS), an airborne lidar scanning laser altimeter. The data were collected as part of a NASA campaign, in collaboration with the European Space Agency (ESA) mission AfriSAR. This collection has `AFLVIS1B` as it's short name.

2.10.21 Searching the Collection

[29]:
```
aflvis1b_collections = maap.searchCollection(
    short_name='AFLVIS1B',
    version='1',
    cmr_host='cmr.earthdata.nasa.gov'
)
aflvis1b_collections
```

[29]:
```
[{'concept-id': 'C1549378019-NSIDC_ECS',
  'revision-id': '43',
  'format': 'application/echo10+xml',
  'Collection': {'ShortName': 'AFLVIS1B',
   'VersionId': '1',
   'InsertTime': '2023-08-22T16:12:12.094Z',
   'LastUpdate': '2023-08-22T16:12:12.094Z',
```

(continues on next page)

(continued from previous page)

```
  'LongName': 'Not provided',
  'DataSetId': 'AfriSAR LVIS L1B Geolocated Return Energy Waveforms V001',
  'Description': 'This data set contains return energy waveform data over Gabon, Africa.
↪ The measurements were taken by the NASA Land, Vegetation, and Ice Sensor (LVIS), an␣
↪airborne lidar scanning laser altimeter. The data were collected as part of a NASA␣
↪campaign, in collaboration with the European Space Agency (ESA) mission AfriSAR.',
  'DOI': {'DOI': '10.5067/ED5IYGVTB50Z'},
  'StandardProduct': 'false',
  'RevisionDate': '2023-05-31T00:00:00.000Z',
  'SuggestedUsage': 'Scientific Research',
  'ProcessingCenter': 'NASA/GSFC/SED/ESD/LRSL',
  'ProcessingLevelId': 'Level 1B',
  'ProcessingLevelDescription': 'Sensor units',
  'ArchiveCenter': 'NASA NSIDC DAAC',
  'CollectionState': 'PLANNED',
  'RestrictionComment': ' These data are freely, openly, and fully accessible, provided␣
↪that you are logged into your NASA Earthdata profile (https://urs.earthdata.nasa.gov/).
↪',
  'UseConstraints': {'LicenseText': ' These data are freely, openly, and fully␣
↪available to use without restrictions, provided that you cite the data according to␣
↪the recommended citation at https://nsidc.org/about/use_copyright.html. For more␣
↪information on the NASA EOSDIS Data Use Policy, see https://earthdata.nasa.gov/earth-
↪observation-data/data-use-policy.'},
  'DataFormat': 'HDF5',
  'SpatialKeywords': {'Keyword': 'GABON'},
  'Temporal': {'EndsAtPresentFlag': 'false',
   'RangeDateTime': {'BeginningDateTime': '2016-02-20T00:00:00.000Z',
    'EndingDateTime': '2016-03-08T23:59:59.999Z'}},
  'Contacts': {'Contact': [{'Role': 'ARCHIVER',
     'OrganizationName': 'NASA NSIDC DAAC',
     'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
       'City': 'Boulder',
       'StateProvince': 'CO',
       'PostalCode': '80309-0449',
       'Country': 'USA'}},
     'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
       'Type': 'Telephone'}},
     'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
    {'Role': 'DISTRIBUTOR',
     'OrganizationName': 'NASA NSIDC DAAC',
     'OrganizationAddresses': {'Address': {'StreetAddress': 'National Snow and Ice Data␣
↪Center CIRES, 449 UCB University of Colorado',
       'City': 'Boulder',
       'StateProvince': 'CO',
       'PostalCode': '80309-0449',
       'Country': 'USA'}},
     'OrganizationPhones': {'Phone': {'Number': '1 303 492 6199',
       'Type': 'Telephone'}},
     'OrganizationEmails': {'Email': 'nsidc@nsidc.org'}},
    {'Role': 'PROCESSOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
    {'Role': 'ORIGINATOR', 'OrganizationName': 'NASA/GSFC/SED/ESD/LRSL'},
```

(continues on next page)

(continued from previous page)

```
    {'Role': 'TECHNICAL CONTACT',
     'ContactPersons': {'ContactPerson': [{'FirstName': 'J.',
        'MiddleName': 'Bryan',
        'LastName': 'Blair',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'Michelle',
        'LastName': 'Hofton',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'NSIDC',
        'MiddleName': 'User',
        'LastName': 'Services',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'J.',
        'MiddleName': 'Bryan',
        'LastName': 'Blair',
        'JobPosition': 'TECHNICAL CONTACT'},
       {'FirstName': 'Michelle',
        'LastName': 'Hofton',
        'JobPosition': 'TECHNICAL CONTACT'}]}}]},
  'ScienceKeywords': {'ScienceKeyword': {'CategoryKeyword': 'EARTH SCIENCE',
    'TopicKeyword': 'SPECTRAL/ENGINEERING',
    'TermKeyword': 'INFRARED WAVELENGTHS',
    'VariableLevel1Keyword': {'Value': 'SENSOR COUNTS'}}},
  'Platforms': {'Platform': [{'ShortName': 'AIRCRAFT',
     'LongName': 'Not provided',
     'Type': 'Aircraft',
     'Instruments': {'Instrument': [{'ShortName': 'ALTIMETERS',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'ALTIMETERS'}}},
       {'ShortName': 'LASERS',
        'LongName': 'Light Amplification by Stimulated Emission of Radiation',
        'Technique': 'instrument',
        'NumberOfSensors': '1',
        'Sensors': {'Sensor': {'ShortName': 'LASERS',
          'LongName': 'Light Amplification by Stimulated Emission of Radiation'}}}]}},
    {'ShortName': 'B-200',
     'LongName': 'Beechcraft King Air B-200',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'C-130',
     'LongName': 'Lockheed C-130 Hercules',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
```

(continues on next page)

(continued from previous page)

```
        'Sensors': {'Sensor': {'ShortName': 'LVIS',
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}},
    {'ShortName': 'DC-8',
     'LongName': 'Douglas DC-8',
     'Type': 'Aircraft',
     'Characteristics': {'Characteristic': {'Name': 'AircraftID',
       'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft',
       'DataType': 'STRING',
       'Unit': 'Not Applicable',
       'Value': 'N817NA'}},
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'G-V',
     'LongName': 'Gulfstream V',
     'Type': 'Jet',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'HU-25C',
     'LongName': 'Dassault HU-25C Guardian',
     'Type': 'Jet',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'P-3B',
     'LongName': 'Lockheed P-3B Orion',
     'Type': 'Propeller',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
         'LongName': 'Land, Vegetation, and Ice Sensor'}}}}},
    {'ShortName': 'RQ-4',
     'LongName': 'Northrop Grumman RQ-4 Global Hawk',
     'Type': 'Uncrewed Aerial Vehicles',
     'Instruments': {'Instrument': {'ShortName': 'LVIS',
       'LongName': 'Land, Vegetation, and Ice Sensor',
       'Technique': 'instrument',
       'NumberOfSensors': '1',
       'Sensors': {'Sensor': {'ShortName': 'LVIS',
```

(continues on next page)

(continued from previous page)

```
          'LongName': 'Land, Vegetation, and Ice Sensor'}}}}}]},
  'AdditionalAttributes': {'AdditionalAttribute': [{'Name': 'AircraftID',
     'DataType': 'STRING',
     'Description': 'The identifier of the airplane used by the FAA to uniquely␣
↪identify each aircraft'},
    {'Name': 'SIPSMetGenVersion',
     'DataType': 'STRING',
     'Description': 'The version of the SIPSMetGen software used to produce the␣
↪metadata file for this granule'},
    {'Name': 'ThemeID',
     'DataType': 'STRING',
     'Description': 'The identifier of the theme under which data are logically grouped
↪'},
    {'Name': 'identifier_product_doi',
     'DataType': 'STRING',
     'Description': 'Digital object identifier that uniquely identifies this data␣
↪product'},
    {'Name': 'identifier_product_doi_authority',
     'DataType': 'STRING',
     'Description': 'URL of the digital object identifier resolving authority'}]},
  'Campaigns': {'Campaign': {'ShortName': 'AfriSAR',
    'LongName': 'NASA campaign in collaboration with European Space Agency (ESA) in␣
↪Gabon, Africa ',
    'StartDate': '2016-02-02T00:00:00.000Z',
    'EndDate': '2016-03-08T00:00:00.000Z'}},
  'SpatialInfo': {'SpatialCoverageType': 'HORIZONTAL'},
  'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://n5eil01u.ecs.nsidc.org/
↪ICEBRIDGE/AFLVIS1B.001',
     'URLDescription': 'Direct download via HTTPS protocol.'},
    {'URL': 'https://search.earthdata.nasa.gov/search?q=AFLVIS1B+V001',
     'URLDescription': "NASA's newest search and order tool for subsetting,␣
↪reprojecting, and reformatting data."},
    {'URL': 'https://nsidc.org/data/data-access-tool/AFLVIS1B/versions/1/',
     'URLDescription': 'Search and filter data files using a map-based interface'}]},
  'OnlineResources': {'OnlineResource': [{'URL': 'https://doi.org/10.5067/ED5IYGVTB50Z',
     'Description': 'Provides access to data, documentation, tools, citation␣
↪information, support, and other resources.',
     'Type': 'CollectionURL : DATA SET LANDING PAGE'},
    {'URL': 'https://doi.org/10.5067/ED5IYGVTB50Z',
     'Description': "Includes a user's guide, supplemental documents like ATBDs and␣
↪academic papers, How Tos, FAQs, etc.",
     'Type': 'VIEW RELATED INFORMATION : GENERAL DOCUMENTATION'}]},
  'Spatial': {'SpatialCoverageType': 'HORIZONTAL',
   'HorizontalSpatialDomain': {'Geometry': {'CoordinateSystem': 'CARTESIAN',
     'BoundingRectangle': {'WestBoundingCoordinate': '8.0',
      'NorthBoundingCoordinate': '1.0',
      'EastBoundingCoordinate': '12.0',
      'SouthBoundingCoordinate': '-2.0'}}},
   'GranuleSpatialRepresentation': 'GEODETIC'}}}]
```

2.10.22 Searching and Downloading a Granule

We will be following the same steps as described above for the other data products.

[31]:
```
COLLECTION_ID = aflvis1b_collections[0]['concept-id']

results = maap.searchGranule(
    concept_id=COLLECTION_ID,
    cmr_host="cmr.earthdata.nasa.gov"
)
pprint(f'Got {len(results)} results')


#Validating download
filename = results[0].getData(dataDir)
print(filename)
```

```
'Got 20 results'
./data/LVIS1B_Gabon2016_0220_R1808_038024.h5
```

2.10.23 Validating the Product

We observe that the downloaded product is stored as a `.h5` file similar to the `ABLVIS1B` and `LVISF1B` data products discussed above. So, we will be using the `h5py` package to validate the dowloaded product.

[32]:
```
#Testing the HDF5 files
import h5py
with h5py.File(filename, "r") as f:
    print("Keys: %s" % f.keys())
    # get first object name/key; may or may NOT be a group
    a_group_key = list(f.keys())[0]

    # get the object type for a_group_key: usually group or dataset
    print(type(f[a_group_key]))

    # If a_group_key is a group name,
    # this gets the object names in the group and returns as a list
    data = list(f[a_group_key])

    # If a_group_key is a dataset name,
    # this gets the dataset values and returns as a list
    data = list(f[a_group_key])
    # preferred methods to get dataset values:
    ds_obj = f[a_group_key]      # returns as a h5py dataset object
    ds_arr = f[a_group_key][()]  # returns as a numpy array

    print(ds_arr)
```

```
Keys: <KeysViewHDF5 ['AZIMUTH', 'INCIDENTANGLE', 'LAT0', 'LAT1023', 'LFID', 'LON0',
↪'LON1023', 'RANGE', 'RXWAVE', 'SHOTNUMBER', 'SIGMEAN', 'TIME', 'TXWAVE', 'Z0', 'Z1023',
↪ 'ancillary_data']>
<class 'h5py._hl.dataset.Dataset'>
[170.88405 171.3952  179.33963 ... 178.35054 178.84067 179.31787]
```

As observed from the output of the above cell, the data values are retrieved from the downloaded product. Hence, validating the downloaded file.

2.10.24 References

1. Blair, J. B. and M. Hofton. (2020). LVIS Facility L1B Geolocated Return Energy Waveforms, Version 1 [Data Set]. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center. https://doi.org/10.5067/XQJ8PN8FTIDG. Date Accessed 08-17-2023.
2. Blair, J. B. and M. Hofton. (2018). ABoVE LVIS L1B Geolocated Return Energy Waveforms, Version 1 [Data Set]. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center. https://doi.org/10.5067/UMRAWS57QAFU. Date Accessed 08-17-2023.
3. Blair, J. B. and M. Hofton. (2020). LVIS Facility L2 Geolocated Surface Elevation and Canopy Height Product, Version 1 [Data Set]. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center. https://doi.org/10.5067/VP7J20HJQISD. Date Accessed 08-17-2023.
4. Blair, J. B. and M. Hofton. (2018). ABoVE LVIS L2 Geolocated Surface Elevation Product, Version 1 [Data Set]. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center. https://doi.org/10.5067/IA5WAX7K3YGY. Date Accessed 08-17-2023.
5. Blair, J. B. and M. Hofton. (2018). AfriSAR LVIS L1B Geolocated Return Energy Waveforms, Version 1 [Data Set]. Boulder, Colorado USA. NASA National Snow and Ice Data Center Distributed Active Archive Center. https://doi.org/10.5067/ED5IYGVTB50Z. Date Accessed 08-17-2023.

2.11 ESA CCI v4 Access and Visualize

Authors: Emile Tenezakis (DevSeed), Rajat Shinde (UAH)

Date: October 2, 2023

Description: In this tutorial, we explore accessing and visualizing ESA CCI Version 4 data from the MAAP STAC catalog. We make use of the stackstac package that allows us to turn a stack collection imported from the catalog with the `pystac_client <https://pystac-client.readthedocs.io/en/stable/>`__ to an xarray dataset, and we plot the time series of the mean aboveground biomass for a selected tile of the dataset across the available temporal range.

2.11.1 Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

2.11.2 About The Dataset

This dataset comprises estimates of forest Above-Ground Biomass (AGB) for the years 2017, 2018, 2019 and 2020, version 4. They are derived from a combination of Earth Observation (EO) data, depending on the year, from the Copernicus Sentinel-1 mission, Envisat's ASAR instrument and JAXA's Advanced Land Observing Satellite (ALOS-1 and ALOS-2), along with additional information from Earth observation sources. The data has been produced as part of the European Space Agency's (ESA's) Climate Change Initiative (CCI) programme by the Biomass CCI team.

2.11.3 Additional Resources

- ESA's Climate Change Initiative Biomass project
- xarray Documentation

2.11.4 Importing and Installing Packages

First off, we will install and import the required Python packages if they are not already installed in the workspace.

[1]:
```
# !mamba install -y -c conda-forge stackstac
# !pip install pystac_client
```

[2]:
```
from stackstac import stack, mosaic
import pystac_client
```

2.11.5 Accessing and Filtering the Items

After installing the require packages, we create a client to access the STAC test catalog.

[3]:
```
URL = "https://stac.maap-project.org"
catalog = pystac_client.Client.open(URL)
```

2.11.6 Creating an AOI

Now, we define a bounding box of interest to find the tile that covers a small region around Manaus, Brazil (Amazon rainforest).

[4]:
```
# BBox for filtering the items in the collection
bbox = [-55,-6,-54.8,-5.8]
```

We proceed to an item search in the catalog using the `pystac-client`, filtering items covering our area of interest.

[5]:
```
stac_collection = catalog.search(
    collections=["ESACCI_Biomass_L4_AGB_V4_100m"],
    bbox=bbox
)
```

Let's take a quick look at the results of the search. We can access the link of the location where the data is stored using `href` attribute as shown below.

[6]:
```
stac_collection.get_all_items()[0].assets['estimates'].href
```

```
/opt/conda/lib/python3.10/site-packages/pystac_client/item_search.py:850: FutureWarning:␣
↪get_all_items() is deprecated, use item_collection() instead.
  warnings.warn(
```

[6]:
```
's3://nasa-maap-data-store/file-staging/nasa-map/ESACCI_Biomass_L4_AGB_V4_100m_2020/
↪N00W060_ESACCI-BIOMASS-L4-AGB-MERGED-100m-2020-fv4.0.tif'
```

We turn the resulting set of items (we expect one item here in the result, given the size of our bounding box) into an xarray DataArray using `stackstac`.

[7]:
```
#Creating a stack of the filtered items
arr = stack(stac_collection.get_all_items())
```

```
/opt/conda/lib/python3.10/site-packages/stackstac/prepare.py:408: UserWarning: The␣
↪argument 'infer_datetime_format' is deprecated and will be removed in a future version.
↪ A strict version of it is now the default, see https://pandas.pydata.org/pdeps/0004-
↪consistent-to-datetime-parsing.html. You can safely remove this argument.
  times = pd.to_datetime(
```

We can see that our array has the following dimensions : `time` (we have four data points, one for each year - 2017, 2018, 2019 and 2020), `latitude`, `longitude` and `band` (we have two bands : `estimates` with the AGB estimate values, and `std_dev` storing the AGB standard deviation values).

[8]:
```
arr
```

[8]:
```
<xarray.DataArray 'stackstac-5517ee4c53a7b2c13f76bf727c00d1fc' (time: 4,
                                                                band: 2,
                                                                y: 11250,
                                                                x: 11251)>
dask.array<fetch_raster_window, shape=(4, 2, 11250, 11251), dtype=float64, chunksize=(1,␣
↪1, 1024, 1024), chunktype=numpy.ndarray>
Coordinates: (12/13)
  * time            (time) datetime64[ns] 2017-01-01 2018-01-01 ... 2020-01-01
    id              (time) <U52 'N00W060_ESACCI-BIOMASS-L4-AGB-MERGED-100m-20...
  * band            (band) <U18 'estimates' 'standard_deviation'
  * x               (x) float64 -60.0 -60.0 -60.0 -60.0 ... -50.0 -50.0 -50.0
  * y               (y) float64 0.0 -0.0008889 -0.001778 ... -9.998 -9.999
    proj:epsg       int64 4326
    ...              ...
    proj:transform  object {0.0, 1.0, -60.0, -0.000888888888888, 0.000888888888...
    proj:shape      object {11250}
    proj:geometry   object {'type': 'Polygon', 'coordinates': [[[-60.0, -9.99...
    title           (band) <U49 'Cloud Optimized GeoTIFF of AGB estimates' 'C...
    description     (band) <U49 'Cloud Optimized GeoTIFF of AGB estimates' 'C...
    epsg            int64 4326
Attributes:
    spec:        RasterSpec(epsg=4326, bounds=(-60.00088888828888, -9.9999999...
    crs:         epsg:4326
    transform:   | 0.00, 0.00,-60.00|\n| 0.00,-0.00, 0.00|\n| 0.00, 0.00, 1.00|
    resolution:  0.00088888888888
```

Now, we will perform a spatial sub-setting operation to select a subset of the created stack based on the user input for running our experiments.

[9]:
```
min_x = -55
max_y = -5.8
max_x = -54.8
min_y = -6.0

arr_sub = arr.sel(x=slice(min_x, max_x), y=slice(max_y, min_y))
arr_sub
```

[9]:
```
<xarray.DataArray 'stackstac-5517ee4c53a7b2c13f76bf727c00d1fc' (time: 4,
                                                                band: 2,
                                                                y: 225, x: 225)>
dask.array<getitem, shape=(4, 2, 225, 225), dtype=float64, chunksize=(1, 1, 225, 225),␣
↪chunktype=numpy.ndarray>
Coordinates: (12/13)
  * time            (time) datetime64[ns] 2017-01-01 2018-01-01 ... 2020-01-01
    id              (time) <U52 'N00W060_ESACCI-BIOMASS-L4-AGB-MERGED-100m-20...
  * band            (band) <U18 'estimates' 'standard_deviation'
  * x               (x) float64 -55.0 -55.0 -55.0 -55.0 ... -54.8 -54.8 -54.8
  * y               (y) float64 -5.801 -5.802 -5.803 ... -5.998 -5.999 -6.0
    proj:epsg       int64 4326
    ...              ...
    proj:transform  object {0.0, 1.0, -60.0, -0.00088888888888, 0.00088888888...
    proj:shape      object {11250}
    proj:geometry   object {'type': 'Polygon', 'coordinates': [[[-60.0, -9.99...
    title           (band) <U49 'Cloud Optimized GeoTIFF of AGB estimates' 'C...
    description     (band) <U49 'Cloud Optimized GeoTIFF of AGB estimates' 'C...
    epsg            int64 4326
Attributes:
    spec:        RasterSpec(epsg=4326, bounds=(-60.00088888828888, -9.9999999...
    crs:         epsg:4326
    transform:   | 0.00, 0.00,-60.00|\n| 0.00,-0.00, 0.00|\n| 0.00, 0.00, 1.00|
    resolution:  0.00088888888888
```

2.11.7 Accessing the Subset for further Processing

As it is evident from the above output that the size of the subset xarray has reduced significantly and it can be used further for our analysis. In this tutorial, we are going to perform following two operations:

1. Masking some values based on a condition
2. Plotting Mean time-series for the AGB estimates

Masking Values Based on a Condition

Below, we will explore how to mask few values based on a condition and plot the time series for the filtered array.

[10]:
```
arr_est = arr_sub.sel(band='estimates')
```

Now, we are going to filter the values which are less than 10 (and keep values greater than 10) using the DataArray.where function. The `.where(<cond>, <True directive>, <False directive>)` function checks a condition and performs specific directives if the condition results in `True` or `False`. In the resulting xarray, all the values less than 10 are going to be `nan` (the default value used if the filtering condition is `True`). We can also replace `nan` by any other value by mentioning it in the `<True directive>`.

[11]:
```
arr_masked_lt = arr_est.where(arr_est >= 10)
arr_masked_lt.values
```

[11]:
```
array([[[222., 132., 193., ..., 360., 246., 271.],
        [153., 113., 152., ..., 331., 180., 261.],
        [189., 112., 173., ..., 381., 269., 260.],
        ...,
        [231., 193., 310., ..., 145., 392., 392.],
        [333., 255., 297., ..., 133., 392., 348.],
        [370., 189., 266., ..., 251., 293., 338.]],

       [[219., 100., 138., ..., 363., 271., 198.],
        [188., 114., 101., ..., 345., 198., 254.],
        [142., 144., 179., ..., 325., 306., 372.],
        ...,
        [353., 302., 379., ..., 162., 385., 385.],
        [332., 348., 321., ..., 151., 385., 314.],
        [375., 243., 283., ..., 201., 336., 342.]],

       [[271., 255., 219., ..., 363., 278., 286.],
        [253., 321., 116., ..., 226., 333., 233.],
        [222., 277., 176., ..., 169., 355., 190.],
        ...,
        [304., 336., 275., ..., 318., 308., 234.],
        [340., 234., 348., ..., 315., 356., 330.],
        [243., 323., 264., ..., 288., 307., 318.]],

       [[157., 217., 171., ..., 206., 161., 253.],
        [190., 229., 226., ..., 214., 210., 314.],
        [269., 212., 152., ..., 268., 402., 324.],
        ...,
        [247., 292., 246., ..., 373., 321., 159.],
        [329., 200., 318., ..., 278., 284., 255.],
        [237., 361., 238., ..., 245., 292., 316.]]])
```

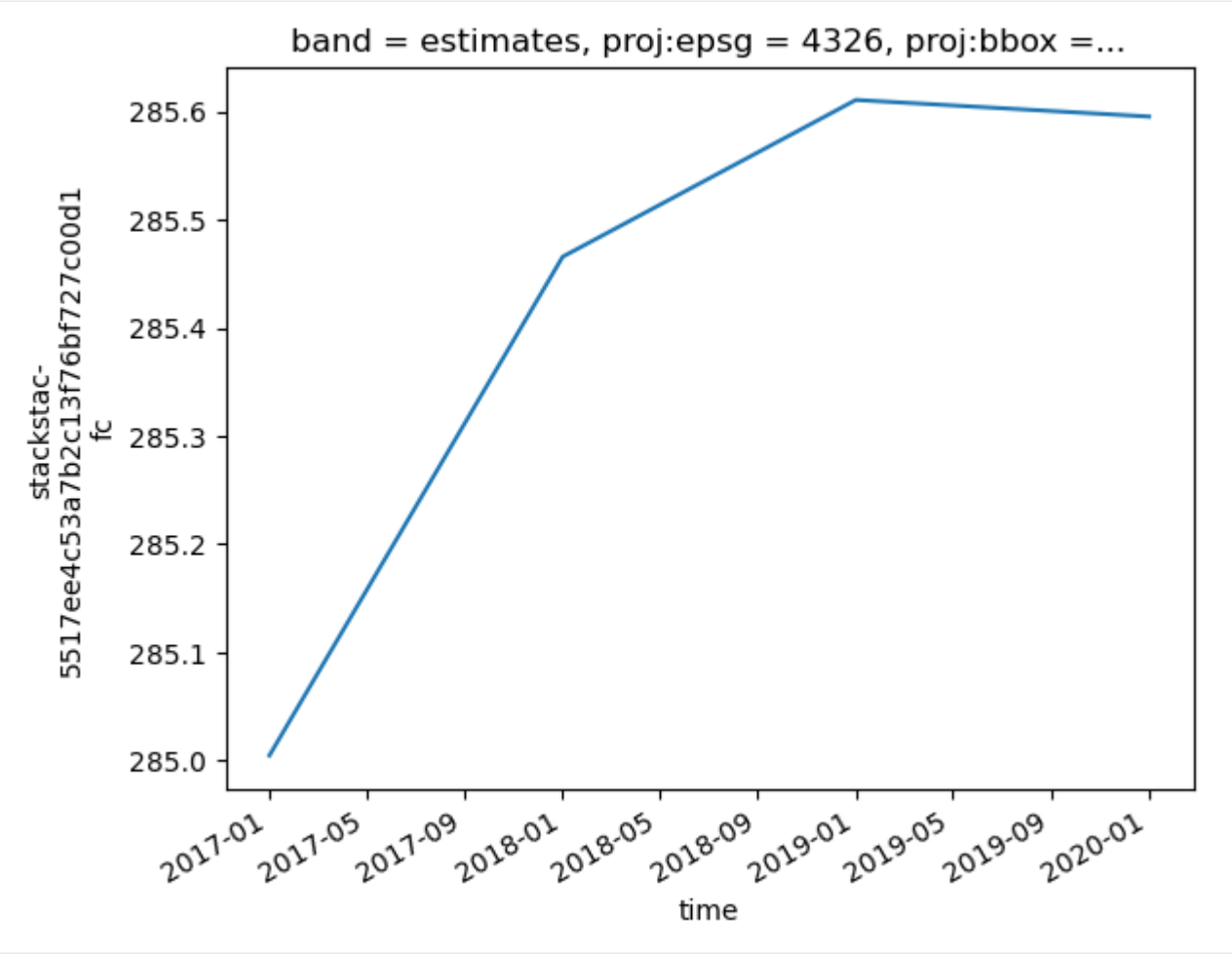
Similarly, we could also show filtering values greater than 10 (and keeping values less than 10) based on above approach and updating the condition to `arr_masked_lt = arr_est.where(arr_est <= 10)`.

Plotting Mean Time-Series for the AGB Estimates

Below we plot the mean time-series of the masked array.

[12]:
```
arr_masked_lt.mean(['x','y']).plot()
```

[12]:
```
[<matplotlib.lines.Line2D at 0x7fd668b54e80>]
```


band = estimates, proj:epsg = 4326, proj:bbox =...
stackstac-
5517ee4c53a7b2c13f76bf727c00d1
fc
285.6
285.5
285.4
285.3
285.2
285.1
285.0
2017-01
2017-05
2017-09
2018-01
2018-05
2018-09
2019-01
2019-05
2019-09
2020-01
time

CHAPTER

THREE

TECHNICAL TUTORIALS

3.1 Search

MAAP users are advised to use two catalogs:

1. Use NASA's Operational CMR to discover NASA-produced and curated data: https://cmr.earthdata.nasa.gov/search/site/docs/search/api.html.
2. Use MAAP STAC for data not found in NASA CMR, and data produced by MAAP users: https://stac.maap-project.org/api.html.

Warning: The https://cmr.maap-project.org catalog was deprecated on **May 1, 2023**. Users should request collections they need from this catalog to be made discoverable in the MAAP STAC or NASA's Operational CMR if they're not already there.

More information on each catalog and migrating from MAAP's CMR here: MAAP's Dual Catalog.

3.1.1 MAAP's Dual Catalog

MAAP users are advised to use two catalogs:

1. Use NASA's Operational CMR to discover NASA-produced and curated data: https://cmr.earthdata.nasa.gov/search/site/docs/search/api.html.
2. Use MAAP STAC for data not found in NASA CMR, and data produced by MAAP users: https://stac.maap-project.org/api.html.

Warning: The https://cmr.maap-project.org catalog was deprecated on **May 1, 2023**. Users should request collections they need from this catalog to be made discoverable in the MAAP STAC or NASA's Operational CMR if they're not already there.

More information on each catalog and migrating from MAAP's CMR is detailed in the bottom of this page.

MAAP STAC

MAAP STAC (https://stac.maap-project.org) is dedicated to datasets not accessible via NASA's CMR, such as GEDI Cal/Val datasets, ESA datasets, and user-shared data products.

STAC discovery

Users can discover data in MAAP STAC using pystac-client or https://stac-browser.maap-project.org.

API documentation is available here: https://stac.maap-project.org/api.html (will return MAAP STAC results).

The general STAC API spec is here: https://api.stacspec.org/v1.0.0-rc.1/core/.

An example of using pystac-client is included above and in *Searching STAC Documentation*.

Data Access via STAC

Data assets (files) published to STAC have not moved from the S3 bucket `s3://nasa-maap-data-store`. ESA data is accessible via public HTTP access. NASA data in S3 is accessible publicly or via role-based bucket policy access.

Users are encouraged to use common AWS S3 libraries for NASA data access, such as Python's boto3.

Each item should have a "data" asset which includes a URL to the data.

For example, https://stac.maap-project.org/collections/BIOSAR1/items/biosar1_roi_lidar58 includes:

```
"assets": {
  "shx": {
    "href": "https://bmap-catalogue-data.oss.eu-west-0.prod-cloud-ocb.orange-business.
↪com/Campaign_data/biosar1/biosar1_roi_lidar58.shx",
    "type": "application/octet-stream",
    "roles": [
      "data"
    ]
  },
}
```

NASA's Operational CMR

CMR Discovery

Users can discover data NASA's Operational CMR via its publicly accessible API: https://cmr.earthdata.nasa.gov and user interface: https://search.earthdata.nasa.gov.

CMR Search documentation can be found in *Searching Collections* and *Searching Granules* and https://cmr.earthdata.nasa.gov/search/site/docs/search/api.html.

CMR Access

For all NASA MAAP users, access to NASA'S Operational data is provided via a federated access token. Anything that is in NASA's Operational CMR should be accessed via maap-py so that the federated access token can be used. Users can also access data from LPDAAC (and possibly other DAACs in the future) without maap-py since the workspace should have access via a role-based bucket policy on the LPDAAC cloud bucket.

Anyone can access data through Earthdata Login as well.

Find more documentation about how to access data in CMR in the Access section of this documentation.

Migrating from MAAP's CMR

If you're migrating code from using https://cmr.maap-project.org, we're here to help. The documentation below should support migrating to https://cmr.earthdata.nasa.gov and https://stac.maap-project.org. If not, please contact the data team for assistance.

Migration Steps:

1. Identify where your code is using https://cmr.maap-project.org and which datasets are being discovered and accessed.
2. Once you've identified the datasets, use https://search.earthdata.nasa.gov or https://stac-browser.maap-project.org to find out if the dataset is available through NASA's Operational CMR or MAAP's STAC catalog. If you don't see your datasets in one of those places, reach out to the data team so they can prioritize that dataset for publication to MAAP STAC.
3. If the dataset is in NASA's Operational CMR and you're using MAAP's Python library `maap-py` to discover and access data, add the parameter `cmr_host="cmr.earthdata.nasa.gov"` to your `maap.searchCollection` and `maap.searchGranule` function calls. Update the `concept_id` to match the one from NASA's Operational CMR if you're using it to identify a specific collection or granule.
4. If the dataset is in MAAP STAC, use `pystac_client` (https://pystac-client.readthedocs.io/en/stable/) or an HTTP library to call the STAC HTTP API endpoints directly.

Examples:

Example of switching a granule search to NASA's Operational CMR:

The code below discovers granules from the `ABoVE LVIS L2 Geolocated Surface Elevation Product`:

```
COLLECTION_ID = 'C1200125288-NASA_MAAP'
results = maap.searchGranule(concept_id=COLLECTION_ID)
pprint(f'Got {len(results)} results')
```

This dataset exists in NASA's Operational CMR. Using https://search.earthdata.nasa.gov, I discovered the collection's `concept_id` by searching for "ABoVE LVIS L2 Geolocated Surface Elevation Product" and copying the `concept_id` from the URL of the result to modify the code below:

```
COLLECTION_ID = 'C1513105984-NSIDC_ECS'
results = maap.searchGranule(
  cmr_host='cmr.earthdata.nasa.gov',
  concept_id=COLLECTION_ID
```

(continues on next page)

(continued from previous page)

```
)
pprint(f'Got {len(results)} results')
```

Example of switching a granule search to MAAP STAC:

This code discovers granules from the `Landsat 8 Operational Land Imager (OLI) Surface Reflectance Analysis Ready Data (ARD) V1, Peru and Equatorial Western Africa, April 2013-January 2020`.

```
COLLECTION_ID = 'C1200110769-NASA_MAAP'

results = maap.searchGranule(concept_id=COLLECTION_ID)
pprint(f'Got {len(results)} results')
```

You can use https://stac-browser.maap-project.org to find the STAC collection ID for that dataset, which is `Landsat8_SurfaceReflectance`.

```
from pystac_client import Client
URL = 'https://stac.maap-project.org/'
cat = Client.open(URL)
for collection in cat.get_all_collections():
    print(collection)

collection = cat.get_collection('Landsat8_SurfaceReflectance')
items = collection.get_items()
```

3.1.2 Using the NASA Earthdata Search Client Graphical User Interface

Author(s): Samuel Ayers (UAH)

Date: July 27, 2020

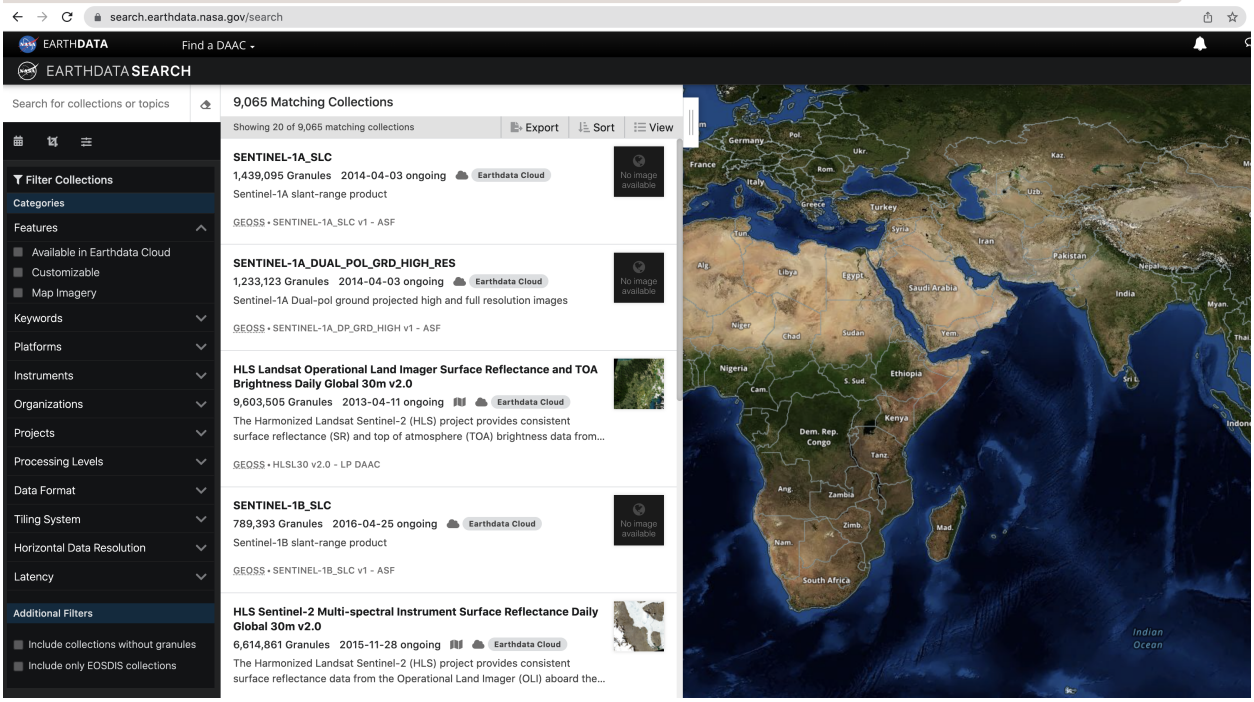
Description: A guide detailing how to use NASA's Earthdata Search client graphical user interface (GUI).

Additional Resources

- Earthdata Search
- Find Earthdata

About the Earthdata Search Client (EDSC)

The EDSC allows users to search, preview, download, and access EOSDIS Earth observation data. NASA's EDSC is located at https://search.earthdata.nasa.gov. ESDC provides a GUI for NASA's Common Metadata Repository (CMR) to ease the process of discovering data.



(*Image of the NASA EDSC GUI*)

Using the Earthdata Search Client

We can use the searchbar to search for a term (example: *GEDI*) to narrow the resulting collections. The search can be further refined by picking a temporal range from the calendar using the **Temporal** button and setting spatial boundaries using the **Spatial** button. Additionally, using the **Advanced search** lets you search by the HUC ID or region (more information about hydrological units may be found here). You can use the **Clear Search** button to undo any filters that have been set. The sidebar to the left allows you to further refine your search by selecting one of the available facets. These include *Features*, *Keywords*, *Platforms*, *Instruments*, *Organizations*, *Projects*, *Processing levels*, *Date Format*, *Tiling System*, *Horizontal Data Resolution*, and *Latency*.

We can also use the tools with the background map to refine our search. We can search spatially using **Search by spatial polygon**, **Search by spatial rectangle**, **Search by spatial circle**, and **Search by spatial coordinate**. Layers may be edited using the **Edit layers** button and deleted using the **Delete layers** buttons. There are also options for **North Polar Stereographic**, **Geographic (Equirectangular)**, and **South Polar Stereographic** projections. There are options to **Zoom in**, **Zoom home**, and **Zoom out**. Finally, we can change the basemap by selecting the **Map layers** button.

The results of the search are displayed in the *Matching Collections* section. Collection names and summaries for each result are shown here. The **View collection details** button may be used to view related URLs and additional information about the selected collection. Also, collections may be added to a project using the **Add collection to the current project** button. Clicking anywhere else on a result allows you to see the granules within the collection available for download.

[ ]:

3.1.3 Searching for Collections in NASA's Operational CMR using maap-py

Authors: Samuel Ayers (UAH), Aimee Barciauskas (DevSeed), Alex Mandel (DevSeed)

Date: November 2, 2020

Description: These examples walk through the MAAP API functionality of searching for collections within NASA's Common Metadata Repository (CMR) based on specific parameters. Collections are groupings of files that share the same product specification. Searching for collections can be useful for finding individual files, known as granules, which are used for processing.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Additional Resources

- NASA's CMR API Documentation

Importing and Installing Packages

We begin by importing the `maap` package and creating a new MAAP class.

[1]:
```
# import the MAAP package to handle queries
from maap.maap import MAAP

# import printing package to help display outputs
from pprint import pprint

# invoke the MAAP search client
maap = MAAP()
```

About searchCollection

We can use the `maap.searchCollection` function to return a list of desired collections. Before using this function, let's use the `help` function to view the specific arguments and keywords for `maap.searchCollection`.

[2]:
```
# view help for the searchCollection function
help(maap.searchCollection)
```

```
Help on method searchCollection in module maap.maap:

searchCollection(limit=100, **kwargs) method of maap.maap.MAAP instance
    Search the CMR collections
    :param limit: limit of the number of results
    :param kwargs: search parameters
    :return: list of results (<Instance of Result>)
```

The help text is showing that `maap.searchCollection` accepts a limit and search parameters. The limit parameter limits the number of resulting collections returned by `maap.searchCollection`. Note that `limit=100` means that the *default limit* for results from the MAAP API is 100. `maap.searchCollection` accepts any additional search parameters that are included in the CMR. For a list of accepted parameters, please refer to the CMR Search Collections API reference.

In this example we will explore search options including:

1. Finding all Collections
2. Searching by temporal filter
3. Searching by spatial filter
4. Using the results from one search as inputs into another
5. Searching by additional attributes

Finding all Collections

Here we will demonstrate how to create a list containing all of the collections contained within the CMR. To do this, we will use the `maap.searchCollection` function without any additional search parameters.

[3]:
```
# search all collections
results = maap.searchCollection(cmr_host="cmr.earthdata.nasa.gov")

# print the number of collections
pprint(f'Got {len(results)} results')
```

```
'Got 100 results'
```

We get 100 results because of the default page limit. The result from the MAAP API is a list of collections where each element in the list is the metadata for that particular collection. To change the limit, type `limit=` and then a value within the parentheses after `maap.searchCollection()`.

Let's look at the metadata for the first collection in our list of results (`results[0]`) using `pprint`. For formatting purposes, we can use the `depth` parameter to control the number of levels of metadata detail to display. By default, there is no constraint on the depth. By setting a `depth` parameter (in this case `depth=2`), we can ensure that the next contained level is replaced by an ellipsis.

[4]:
```
# print the metadata for the first collection
# we use the depth parameter to set the layer of metadata detail in the results, with␣
↪(1) having the least detail
# (1) displays the concept ID, format, and revision ID
# adjust the depth to a larger value (6) if you would like to view all of the metadata
pprint(results[0], depth=2)
```

```
{'Collection': {'AssociatedBrowseImageUrls': {...},
                'CollectionState': 'COMPLETE',
                'Contacts': {...},
                'DOI': {...},
                'DataSetId': "'Latent reserves' within the Swiss NFI",
                'Description': 'The files refer to the data used in Portier et '
                               'al. "u2018Latent reservesu2019: a hidden '
                               'treasure in National Forest Inventories" '
                               '(2020) *Journal of Ecology*.           '
                               "**'Latent reserves'** are defined as plots in "
```

(continues on next page)

(continued from previous page)

```
                        'National Forest Inventories (NFI) that have '
                        'been free of human influence for >40 to >70 '
                        'years. They can be used to investigate and '
                        'acquire a deeper understanding of attributes '
                        'and processes of near-natural forests using '
                        'existing long-term data. To determine which '
                        'NFI sample plots could be considered '
                        'u2018latent reservesu2019, criteria were '
                        'defined based on the information available in '
                        'the Swiss NFI database:           * Shrub '
                        'forests were excluded.  * Plots must have been '
                        'free of any kind of management, including '
                        'salvage logging or sanitary cuts, for a '
                        'minimum amount of time. Thresholds of 40, 50, '
                        '60 and 70 years without intervention were '
                        'tested.  * To ensure that species composition '
                        'was not influenced by past management, plots '
                        'where potential vegetation was classified as '
                        'deciduous by Ellenberg & Klu00f6tzli (1972) '
                        'had to have an observed proportion of '
                        'deciduous trees matching the theoretical '
                        'proportion expected in a natural deciduous '
                        'forest, as defined by Kienast, Brzeziecki, & '
                        'Wildi (1994).  * Plots had to originate from '
                        'natural regeneration.   * Intensive livestock '
                        'grazing must never have occurred on the '
                        'plots.          The tables stored here were '
                        'derived from the first, second and third '
                        'campaigns of the Swiss NFI. The raw data from '
                        'the Swiss NFI can be provided free of charge '
                        'within the scope of a contractual agreement '
                        '(http://www.lfi.ch/dienstleist/daten-en.php).    '
                        "****    The files 'Data figure 2' to 'Data "
                        "figure 8' are publicly available and contain "
                        'the data used to produce the figures published '
                        "in the paper.     The files 'Plot-level data "
                        "for characterisation of 'latent reserves' and "
                        "'Tree-level data for characterisation of "
                        "'latent reserves' contain all the data "
                        'required to reproduce the section of the '
                        'article concerning the characterisation of '
                        "'latent reserves' and the comparison to "
                        "managed forests. The file 'Data for mortality "
                        "analyses' contains the data required to "
                        'reproduce the section of the article '
                        "concerning tree mortality in 'latent "
                        "reserves'. The access to these three files is "
                        'restricted as they contain some raw data from '
                        'the Swiss NFI, submitted to the Swiss law and '
                        'only accessible upon contractual agreement.   ',
        'InsertTime': '2020-07-29T14:18:59.791Z',
        'LastUpdate': '2021-02-04T04:39:30.512Z',
```

(continues on next page)

(continued from previous page)

```
                'LongName': 'Not provided',
                'OnlineResources': {...},
                'Platforms': {...},
                'ProcessingLevelId': 'Not provided',
                'RestrictionComment': 'Access to the data upon request',
                'RevisionDate': '2021-02-04T04:39:30.512Z',
                'ScienceKeywords': {...},
                'ShortName': 'latent-reserves-in-the-swiss-nfi',
                'Spatial': {...},
                'Temporal': {...},
                'UseConstraints': {...},
                'VersionId': '1.0'},
 'concept-id': 'C1931110427-SCIOPS',
 'format': 'application/echo10+xml',
 'revision-id': '6'}
```

The `Collection` key has all of the collection information including attributes, the archive center, spatial, and temporal information. The `concept-id` is a unique identifier for this collection. It can be used to further refine search results from the CMR, such as when searching for granule information.

Searching by Temporal Filter

Here we use a temporal filter to narrow down our results using the `temporal` keyword in our search. The temporal keyword takes datetime information in a specific format. The date format used is `YYYY-MM-DDThh:mm:ssZ` and temporal search criteria may be either a single date or a date range. If one date is provided then it can be inferred as the start or end date. To define a start date and return all collections after the date, put a comma after the date (`YYYY-MM-DDThh:mm:ssZ,`). To define a end date and return all granules prior to the data, put a comma before the date (`,YYYY-MM-DDThh:mm:ssZ`). Lastly, to get a date range, provide the start date and end date separated by a comma (`YYYY-MM-DDThh:mm:ssZ,YYYY-MM-DDThh:mm:ssZ`).

In this example we will search for one month of data.

[5]:
```
datetimeRange = '2000-01-01T00:00:00Z,2000-01-31T23:59:59Z' # specify datetime range to␣
↪search for data in January 2000

results = maap.searchCollection(
    cmr_host = "cmr.earthdata.nasa.gov",
    temporal = datetimeRange
)
pprint(f'Got {len(results)} results')
```

```
'Got 100 results'
```

[6]:
```
collectionName = results[0]['Collection']['ShortName'] # get the collection short name
collectionDate = results[0]['Collection']['Temporal']['RangeDateTime']['BeginningDateTime
↪'] # get the collection start time

pprint(
    f'Collection {collectionName} was acquired starting at {collectionDate}', width=100)
```

```
'Collection ERS-2_L0 was acquired starting at 1995-10-01T03:13:03.000Z'
```

It appears the first result correctly matches with the beginning and ending temporal search parameters. Keep in mind that the results are limited to 100 so the final collection returned may not match the end date that was searched for.

Searching by Spatial Filter

Here we will illustrate how to search for collections by a spatial filter. There are a couple of spatial filters available to search by in the CMR including point, line, polygon, and bounding box. In this example, we will explore filtering with a bounding box which is a sequence of four latitude and longitude values in the order of `[W,S,E,N]`.

[7]:
```
collectionDomain = '-42,10,42,20' # specify bounding box to search by

results = maap.searchCollection(
    cmr_host = "cmr.earthdata.nasa.gov",
    bounding_box = collectionDomain
)
pprint(f'Got {len(results)} results')
```

```
'Got 100 results'
```

[8]:
```
collectionName = results[3]['Collection']['ShortName'] # get a collection short name
collectionGeometry = results[3]['Collection']['Spatial']['HorizontalSpatialDomain'][
↪'Geometry'] # grab the spatial information from collection

pprint(f'Collection {collectionName} was acquired within the following geometry: ',␣
↪width=100)
pprint(collectionGeometry)
```

```
'Collection gov.noaa.nodc:0000029 was acquired within the following geometry: '
{'BoundingRectangle': {'EastBoundingCoordinate': '-16.25',
                       'NorthBoundingCoordinate': '46.263167',
                       'SouthBoundingCoordinate': '0.766667',
                       'WestBoundingCoordinate': '-124.041667'},
 'CoordinateSystem': 'CARTESIAN'}
```

We can see from the first collection that the spatial coordinates of the collection intersect our search box.

3.1.4 Searching for Granules in NASA's Operational CMR using maap-py

Authors: Kel Markert (UAH), Katrina Virts (UAH), Samuel Ayers (UAH), Alex Mandel (DevSeed)

Date: February 27, 2020 (updated in 2022)

Description: These examples will walk through the MAAP API functionality of searching granules within a collection in NASA's Common Metadata Repository (CMR) based on specific parameters. Granules are individual files from a sensor where a group of granules make a collection within CMR. The granules are the raw data that will be used for processing.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Additional Resources

- NASA's CMR API Documentation

Importing and Installing Packages

We begin by importing the `maap` and `pprint` packages. Then invoke the `MAAP` constructor, setting the `maap_host` argument to `'api.maap-project.org'`.

[1]:
```
# import the MAAP package
from maap.maap import MAAP

# import printing package to help display outputs
from pprint import pprint

# invoke the MAAP constructor using the maap_host argument
maap = MAAP(maap_host='api.maap-project.org')
```

About searchGranule

Here we view the specific arguments and keywords for the `maap.searchGranule` function.

[2]:
```
help(maap.searchGranule)
```

```
Help on method searchGranule in module maap.maap:

searchGranule(limit=20, **kwargs) method of maap.maap.MAAP instance
    Search the CMR granules

    :param limit: limit of the number of results
    :param kwargs: search parameters
    :return: list of results (<Instance of Result>)
```

As we can see from the result, `maap.searchGranule` accepts a limit keyword which limits the number of results from CMR. `maap.searchGranule()` also accepts any additional search parameters that are included in CMR. For a list of accepted parameters, please refer to the CMR Search Granules API reference.

It is important to note that *the default limit on results from the MAAP API is 20*. To increase the number of results we will specify a variable and use it in later queries.

[3]:
```
# get at max 500 results from CMR
MAX_RESULTS = 500
```

In this example we will explore search options including:

1. Searching by collection concept ID
2. Searching by temporal filter
3. Searching by spatial filter
4. Using the results from one search as inputs into another
5. Searching by additional attributes

For the next couple of examples, we will focus on the ICESat-2/ATLAS Land and Vegetation Height dataset.

Searching by Collection Short Name, Version

Here we will search by a short name and version which should uniquely identify a collection CMR. HOWEVER, some datasets exist both in the cloud and on-prem, so in the following example we actually get **2** results.

[4]:
```
atl08_collections = maap.searchCollection(
    short_name='ATL08',
    version='005',
    cmr_host='cmr.earthdata.nasa.gov'
)
len(atl08_collections)
```

[4]:
```
2
```

If you inspect the results, you will see the second result has distribution information which points to an S3 bucket location. You can see this information with the follow code: `atl08_collections[1]['Collection']['DirectDistributionInformation']`.

A simpler solution to finding just the cloud-hosted dataset is to add the `cloud_hosted="true"` parameter to our search.

[5]:
```
atl08_collections = maap.searchCollection(
    short_name='ATL08',
    version='005',
    cmr_host='cmr.earthdata.nasa.gov',
    cloud_hosted="true"
)
len(atl08_collections)
```

[5]:
```
1
```

Now we can look up the collection concept id to find only granules in the cloud-hosted ATL08 v005 dataset.

[6]:
```
COLLECTION_ID = atl08_collections[0]['concept-id']

results = maap.searchGranule(
    concept_id=COLLECTION_ID,
    cmr_host='cmr.earthdata.nasa.gov',
    limit=MAX_RESULTS)
pprint(f'Got {len(results)} results')
```

```
'Got 500 results'
```

We were able to get 500 results! There are most likely more than 500 granules in search results, but remember we limited the results to 500 granules. The result from the MAAP API is a list of granules where each element in the list is the metadata for that particular granule.

Now let's look at the metadata for the first result.

[7]:
```
# print the first granule's metadata
# we use the depth parameter to set the layer of metadata detail in the results, with␣
↪(1) having the least detail
# (1) displays the collection concept ID, concept ID, format, and revision ID
# adjust the depth to a larger value (6) if you would like to view all of the metadata
pprint(results[0], depth=2)
```

```
{'Granule': {'AssociatedBrowseImageUrls': {...},
             'Collection': {...},
             'DataGranule': {...},
             'GranuleUR': 'ATL08_20181014001049_02350102_005_01.h5',
             'InsertTime': '2021-11-14T23:43:07.741Z',
             'LastUpdate': '2021-11-14T23:43:07.741Z',
             'OnlineAccessURLs': {...},
             'OnlineResources': {...},
             'OrbitCalculatedSpatialDomains': {...},
             'Spatial': {...},
             'Temporal': {...}},
 'collection-concept-id': 'C2153574670-NSIDC_CPRD',
 'concept-id': 'G2166182816-NSIDC_CPRD',
 'format': 'application/echo10+xml',
 'revision-id': '1'}
```

There is a lot of information in the metadata so let's break it down…

The `Granule` key has all of the granule information including attributes, browse imagery URLs, spatial, and temporal information. The `collection-concept-id` should match what you searched by and be the same for each granule. Lastly the granule specific `concept-id` is a unique identifier for this granule. This information can be used to further refine search results from CMR, specifically the granule information.

Searching by Temporal Filter

Here we will combine a search from earlier using the Collection Concept ID with a temporal filter to fine tune our results using the `temporal` keyword in our search.

The temporal keyword takes datetime information in a specific format. The date format used is `YYYY-MM-DDThh:mm:ssZ` and temporal search criteria may be either a single date or a date range. If one date is provided then it can be inferred as start or end date. To define a start date and return all granules after the date, put a comma after the date (`YYYY-MM-DDThh:mm:ssZ,`). To define an end date and return all granules prior to the data, put a comma before the date (`,YYYY-MM-DDThh:mm:ssZ`). Lastly, to get a date range, provide the start date and end date separated by a comma (`YYYY-MM-DDThh:mm:ssZ,YYYY-MM-DDThh:mm:ssZ`).

In this example we will search for one month of data.

[8]:
```
date_range = '2018-12-01T00:00:00Z,2018-12-31T23:59:59Z' # specify a date range to␣
↪search for data for Dec. 2018

results = maap.searchGranule(
    temporal=date_range,
    concept_id=COLLECTION_ID,
    limit=MAX_RESULTS,
    cmr_host="cmr.earthdata.nasa.gov"
```

(continues on next page)

(continued from previous page)

```
)
pprint(f'Got {len(results)} results')
```

```
'Got 500 results'
```

[9]:
```
granuleFilename = results[0]['Granule']['DataGranule']['ProducerGranuleId'] # get the granule file name
granuleDate = results[0]['Granule']['Temporal']['RangeDateTime']['BeginningDateTime'] # get the granule start time

pprint(f'Granule {granuleFilename} was acquired starting at {granuleDate}',width=100)
```

```
'Granule ATL08_20181201001339_09680103_005_01.h5 was acquired starting at 2018-12-01T00:13:48.477Z'
```

It looks like the first result correctly matches with the beginning temporal search parameter. Keep in mind that the results are limited to 500 so the final granule returned may not match the end date that was searched for.

Searching by Spatial Filter

Here we will illustrate how to search for granules by a spatial filter. There are a couple of spatial filters available to search by in CMR including point, line, polygon, and bounding box. The most simple to use is the bounding box which is a sequence of four latitude and longitude values in the order of `[W,S,E,N]`. In this example, we are going to search for data over Gabon using the `bounding_box` keyword.

[10]:
```
granule_bbox = '8.79799563969,-3.97882659263,14.4254557634,2.32675751384' # specify bounding box to search by

COLLECTION_ID = 'C1000000240-LPDAAC_ECS' # Collection title: "NASA Shuttle Radar Topography Mission Global 1 arc second V003"

results = maap.searchGranule(
    concept_id=COLLECTION_ID,
    bounding_box=granule_bbox,
    cmr_host="cmr.earthdata.nasa.gov"
)
pprint(f'Got {len(results)} results')
```

```
'Got 20 results'
```

[11]:
```
granule_filename = results[0]['Granule']['DataGranule']['ProducerGranuleId'] # get the granule file name
geometry = results[0]['Granule']['Spatial']['HorizontalSpatialDomain']['Geometry'] # grab the spatial information from granule

pprint(f'Granule {granule_filename} was acquired within the following geometry: ', width=100)
pprint(geometry)
```

```
'Granule S03E012.SRTMGL1.hgt.zip was acquired within the following geometry: '
{'BoundingRectangle': {'EastBoundingCoordinate': '13.00027778',
                       'NorthBoundingCoordinate': '-1.99972222',
```

(continues on next page)

(continued from previous page)

```
                        'SouthBoundingCoordinate': '-3.00027778',
                        'WestBoundingCoordinate': '11.99972222'}}
```

We can see from the first granule that the spatial coordinates of the granule intersect our search box.

The MAAP API provides rich functionality to interact with the CMR instance within the MAAP platform. Users can search datasets programmatically by many parameters and even combine parameters such as spatial and temporal filters to refine results.

Generating ID List from Search Results

Each element in the `results` list contains the metadata for the granules returned by the search. Within this metadata is the key concept-id, which is the unique identifier for each granule. To create a list of granule IDs, we create a new list and add the concept-id from each element of results into the that list.

[12]:
```
granuleID_list = [result['concept-id'] for result in results]

# View some of the results
granuleID_list[:10]
```

[12]:
```
['G1004577874-LPDAAC_ECS',
 'G1004578009-LPDAAC_ECS',
 'G1004578073-LPDAAC_ECS',
 'G1004578089-LPDAAC_ECS',
 'G1004578257-LPDAAC_ECS',
 'G1004578334-LPDAAC_ECS',
 'G1004578381-LPDAAC_ECS',
 'G1004578586-LPDAAC_ECS',
 'G1004578726-LPDAAC_ECS',
 'G1004578728-LPDAAC_ECS']
```

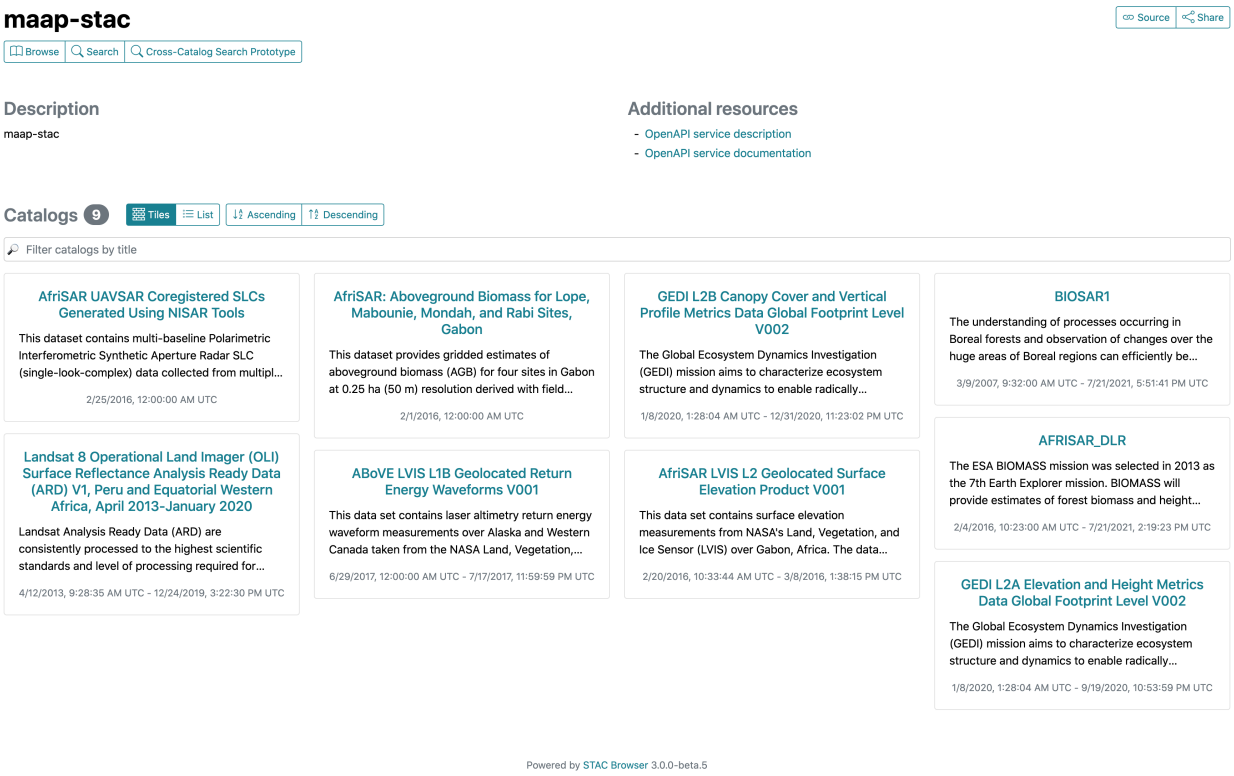
3.1.5 Searching the STAC Catalog

Authors: Aimee Barciauskas (Development Seed)

Date: December 13, 2022

Description: This tutorial provides a basic introduction to searching the MAAP STAC catalog using `pystac-client`.

Another method of searching the STAC catalog is via the STAC browser.



About the STAC Catalog

The MAAP STAC catalog provides discovery of a subset of MAAP datasets. These collections are hosted specifically through the MAAP STAC catalog and are typically not available on NASA's CMR. The data files have not been moved at all in the process of publishing datasets to STAC.

Data will continue to be added to the STAC catalog with priority given to datasets which are known to be in-use by MAAP UWG members through S3 metrics, direct collaboration with data team members, and by request.

Additional Resources

- Pystac-client Introduction

Importing and Installing Packages

In order to run this notebook you'll need the following packages:

[1]:
```
%%capture
%pip install -U pystac-client
```

[2]:
```
from pystac_client import Client
```

STAC Client

We first connect to an API by retrieving the root catalog, or landing page, of the API with the Client.open function.

[3]:
```
# STAC API root URL
URL = 'https://stac.maap-project.org/'
cat = Client.open(URL)
cat
```

[3]:
```
<Client id=stac-fastapi>
```

Searching Collections

As with a static catalog the get_collections function will iterate through the Collections in the Catalog. Notice that because this is an API it can get all the Collections through a single call, rather than having to fetch each one individually.

[4]:
```
stac_collections = list(cat.get_collections())
stac_collections
```

[4]:
```
[<CollectionClient id=Landsat8_SurfaceReflectance>,
 <CollectionClient id=Global_PALSAR2_PALSAR_FNF>,
 <CollectionClient id=Global_Forest_Change_2000-2017>,
 <CollectionClient id=AFRISAR_DLR2>,
 <CollectionClient id=AfriSAR_UAVSAR_KZ>,
 <CollectionClient id=AfriSAR_UAVSAR_Ungeocoded_Covariance>,
 <CollectionClient id=AfriSAR_UAVSAR_Normalization_Area>,
 <CollectionClient id=AfriSAR_UAVSAR_Geocoded_SLC>,
 <CollectionClient id=AfriSAR_UAVSAR_Geocoded_Covariance>,
 <CollectionClient id=GlobCover_09>,
 <CollectionClient id=GlobCover_05_06>,
 <CollectionClient id=GEDI_CalVal_Field_Data>,
 <CollectionClient id=AfriSAR_UAVSAR_Coreg_SLC>,
 <CollectionClient id=GEDI_CalVal_Lidar_Data_Compressed>,
 <CollectionClient id=GEDI_CalVal_Lidar_Data>,
 <CollectionClient id=ABoVE_UAVSAR_PALSAR>,
 <CollectionClient id=AFRISAR_DLR>,
 <CollectionClient id=BIOSAR1>,
 <CollectionClient id=icesat2-boreal>,
 <CollectionClient id=ICESat2_Boreal_AGB_tindex_average>,
 <CollectionClient id=NCEO_Africa_AGB_100m_2017>,
 <CollectionClient id=Paraguay_Country_Pilot>,
 <CollectionClient id=ESACCI_Biomass_L4_AGB_V4_100m>,
 <CollectionClient id=NASA_JPL_global_agb_mean_2020>,
 <CollectionClient id=SRTMGL1_COD>]
```

[5]:
```
collection = cat.get_collection(stac_collections[0].id)
collection
```

[5]:
```
<CollectionClient id=Landsat8_SurfaceReflectance>
```

Searching STAC Items

Query the `/search` endpoint of the STAC catalog to find items in our collection. This method will return an ItemSearch instance which we can then turn into a list.

Read more about additional parameters to the `search()` method at pystac-client.readthedocs.io.

[6]:
```
collection_items = list(cat.search(collections=[collection.id], max_items=10).items())
collection_items
```

[6]:
```
[<Item id=LC08009066201912401T1-SC20200127151508.tar>,
 <Item id=LC08009065201912401T1-SC20200127151451.tar>,
 <Item id=LC08009064201912401T2-SC20200127163402.tar>,
 <Item id=LC08008066201912101T2-SC20200127163324.tar>,
 <Item id=LC08008065201912101T2-SC20200127163702.tar>,
 <Item id=LC08008064201912101T1-SC20200127162047.tar>,
 <Item id=LC08009066201912001T1-SC20200127162000.tar>,
 <Item id=LC08009065201912001T1-SC20200127151500.tar>,
 <Item id=LC08009064201912001T1-SC20200127163722.tar>,
 <Item id=LC08008066201912001T2-SC20200127161947.tar>]
```

We can get a specific item by supplying one of the IDs from an item in our previous collection search. We are then able to get the HREF of the first asset in our item.

[7]:
```
item = collection.get_item(collection_items[0].id)
item.assets[list(item.assets.keys())[0]].href
```

[7]:
```
's3://nasa-maap-data-store/file-staging/nasa-map/Landsat8_SurfaceReflectance___1/
↪LC08009066201912401T1-SC20200127151508.tar.gz'
```

Here's a simplified example:

[ ]:
```
# Retrieve a specific collection
collection = cat.get_collection("ESACCI_Biomass_L4_AGB_V4_100m")

# Search for items in the collection
collection_items = list(cat.search(collections=["ESACCI_Biomass_L4_AGB_V4_100m"], max_
↪items=10).items())

# Retrieve a specific item
item = collection.get_item("S50W080_ESACCI-BIOMASS-L4-AGB-MERGED-100m-2020-fv4.0")

# List the item's asset href
item.assets["estimates"].href
```

3.2 Visualize

3.2.1 Visualizing MAAP STAC Dataset with MosaicJSON

Authors: Samuel Ayers (UAH), Alex Mandel (DevSeed), Aimee Barciauskas (DevSeed)

Date: July 23, 2021

Description: In this notebook, we visualize SRTM Cloud-Optimized GeoTIFFs (COGs) from MAAP's STAC using a generated mosaic from MAAP's TiTiler.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Note About the Data

The NASA Shuttle Radar Topographic Mission (SRTM) has provided digital elevation data (DEMs) for over 80% of the globe. This data is currently distributed free of charge by USGS and is available for download from the National Map Seamless Data Distribution System, or the USGS FTP site.

At MAAP, we've converted this elevation data into Cloud-Optimized GeoTIFFs (COGs) so they can be efficiently queried and visualized. These COGs are available in the MAAP STAC.

Additional Resources

- USGS EROS Archive - Shuttle Radar Topography Mission (SRTM)
- TiTiler API Documentation - MosaicJSON
- Github - MosaicJSON
- Working with MosaicJSON - TiTiler (DevSeed Example)

Importing and Installing Packages

To be able to run this notebook you'll need the following requirements:

- rasterio
- folium
- requests
- pystac-client
- cogeo-mosaic (Optional)

If the packages below are not installed already, uncomment the following cell:

[1]:
```
# %pip install rasterio
# %pip install folium
# %pip install requests
# %pip install pystac-client
# %pip install cogeo-mosaic --pre
```

[2]:
```
import requests

from pystac_client import Client

from pprint import pprint

from rasterio.features import bounds as featureBounds

from folium import Map, TileLayer, GeoJson
```

Fetch SRTM COG STAC Items

[3]:
```
cat = Client.open('https://stac.maap-project.org/')

items = list(
    cat.search(
        collections="SRTMGL1_COD",
        bbox=[4, 42, 16, 48],
        max_items=120
    ).items_as_dicts(),
)
```

Map the Data Bounds

[4]:
```
geojson = {"type": "FeatureCollection", "features": items}

bounds = featureBounds(geojson)
zoom_start = 5

m = Map(
    tiles="OpenStreetMap",
    location=((bounds[1] + bounds[3]) / 2, (bounds[0] + bounds[2]) / 2),
    zoom_start=zoom_start,
)

geo_json = GeoJson(
    data=geojson,
    style_function=lambda x: {
        "opacity": 1,
        "dashArray": "1",
        "fillOpacity": 0,
        "weight": 1,
    },
```

(continues on next page)

(continued from previous page)

```
)
geo_json.add_to(m)
m
```

[4]:
```
<folium.folium.Map at 0x7f007b1aeef0>
```

Create Mosaic

We're using the TiTiler deployed by MAAP

[5]:
```
titiler_endpoint = "https://titiler.maap-project.org"  # MAAP titiler endpoint
```

To begin, we'll pull our COG:

[6]:
```
cog_info = requests.get(
    f"{titiler_endpoint}/cog/info",
    params={
        "url": geojson["features"][0]["assets"]["cog_default"]["href"],
    },
).json()
pprint(cog_info)
```

```
{'band_descriptions': [['b1', '']],
 'band_metadata': [['b1', {}]],
 'bounds': [15.99986111111111,
            47.999861111111116,
            17.000138888888888,
            49.00013888888889],
 'colorinterp': ['gray'],
 'count': 1,
 'driver': 'GTiff',
 'dtype': 'int16',
 'height': 3601,
 'maxzoom': 12,
 'minzoom': 8,
 'nodata_type': 'Nodata',
 'nodata_value': -32768.0,
 'overviews': [2, 4, 8, 16],
 'width': 3601}
```

Next, we will create the mosaic using the geojson from above. SRTMGL1 COGs have a "cog default" asset type, so we can create a mosaic of these type="image/tiff" assets. We can get access to these assets by using the accessor function.

[7]:
```
from cogeo_mosaic.mosaic import MosaicJSON

mosaicdata = MosaicJSON.from_features(
    geojson.get("features"),
    minzoom=cog_info["minzoom"],
    maxzoom=cog_info["maxzoom"],
    accessor=lambda feature: feature["assets"]["cog_default"]["href"],
)
```

Alternatively, we can build a MosaicJSON using a list of files within MAAP's ADE by converting the local "my-private", "my-public", and "shared-buckets" paths to their respective AWS S3 prefixes like so:

```
from maap.maap import MAAP
maap = MAAP(maap_host='api.maap-project.org')
username = maap.profile.account_info()["username"]

local_path = "/local/path/to/files/"

files = glob.glob(os.path.join(dps_output, "*.tif"), recursive=False)

def local_to_s3(url):
    ''' A Function to convert local paths to s3 urls'''
    if url.startswith("my-private-bucket"):
        return url.replace("my-private-bucket", f"s3://maap-ops-workspace/{username}")

    if url.startswith("my-public-bucket"):
        return url.replace("my-public-bucket", f"s3://maap-ops-workspace/shared/
↪{username}")

    if url.startswith("shared-buckets"):
        return url.replace("shared-buckets", "s3://maap-ops-workspace/shared")

tiles = [local_to_s3(file) for file in files]

mosaicdata = MosaicJSON.from_urls(tiles, minzoom=9, maxzoom=16)
```

Now we will upload the mosaicjson to the TiTiler:

[8]:
```
mosaic_links = requests.post(
    url=f"{titiler_endpoint}/mosaics",
    headers={
        "Content-Type": "application/vnd.titiler.mosaicjson+json",
    },
    json=mosaicdata.model_dump(exclude_none=True),
).json()

pprint(mosaic_links)
```

```
{'id': 'a6739144-b9ab-4ef6-82c3-f092100858cb',
 'links': [{'href': 'https://titiler.maap-project.org/mosaics/a6739144-b9ab-4ef6-82c3-
↪f092100858cb',
            'rel': 'self',
            'title': 'Self',
            'type': 'application/json'},
           {'href': 'https://titiler.maap-project.org/mosaics/a6739144-b9ab-4ef6-82c3-
↪f092100858cb/mosaicjson',
            'rel': 'mosaicjson',
            'title': 'MosaicJSON',
            'type': 'application/json'},
           {'href': 'https://titiler.maap-project.org/mosaics/a6739144-b9ab-4ef6-82c3-
↪f092100858cb/tilejson.json',
            'rel': 'tilejson',
            'title': 'TileJSON',
```

(continues on next page)

(continued from previous page)

```
           'type': 'application/json'},
          {'href': 'https://titiler.maap-project.org/mosaics/a6739144-b9ab-4ef6-82c3-
↪f092100858cb/tiles/{z}/{x}/{y}',
           'rel': 'tiles',
           'title': 'Tiles',
           'type': 'application/json'},
          {'href': 'https://titiler.maap-project.org/mosaics/a6739144-b9ab-4ef6-82c3-
↪f092100858cb/WMTSCapabilities.xml',
           'rel': 'wmts',
           'title': 'WMTS',
           'type': 'application/json'}]}
```

The response from the post request gives endpoints for different services (eg. mosaicjson, tilejson, tiles, wmts, etc). We're fetching the `tilejson` endpoint.

[9]:
```
tilejson_endpoint = list(
    filter(lambda x: x.get("rel") == "tilejson", dict(mosaic_links)["links"])
)
tilejson_endpoint
```

[9]:
```
[{'href': 'https://titiler.maap-project.org/mosaics/a6739144-b9ab-4ef6-82c3-f092100858cb/
↪tilejson.json',
  'rel': 'tilejson',
  'type': 'application/json',
  'title': 'TileJSON'}]
```

Display Tiles

NOTE: The important bit is the "tiles" endpoint returned from `f"{titiler_endpoint}/mosaics`. This endpoint (e.g. `https://titiler.maap-project.org/mosaics/{mosaic_id}/tiles/{z}/{x}/{y}`) could be used in any map client which can tile `x,y,z` layers.

Read more about this endpoint at MAAP's TiTiler docs.

[10]:
```
params = {"colormap_name": "viridis", "rescale": "0,4000"}

r_te = requests.get(tilejson_endpoint[0]["href"], params=params).json()

tiles = TileLayer(tiles=f"{r_te['tiles'][0]}", opacity=1, attr="USGS")

tiles.add_to(m)
m
```

[10]:
```
<folium.folium.Map at 0x7f007b1aeef0>
```

[ ]:

3.2.2 Visualizing rasters with MosaicJSON using stac_ipyleaflet

[ ]:
```
%pip install cogeo_mosaic --quiet
```

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

[2]:
```
# Set variables and helper functions
import os

titiler_url = "https://titiler.maap-project.org"
min_zoom = 0
max_zoom = 0
band_min = 0
band_max = 4000
user = os.getenv('CHE_WORKSPACE_NAMESPACE')
bucket = "maap-ops-workspace"
color_map = "gist_earth_r"
```

[3]:
```
def checkFilePath(file_path):
    result = s3.list_objects(Bucket=bucket, Prefix=file_path)
    exists = True if 'Contents' in result else False
    if exists:
        print('PATH EXISTS')
        return [i for i in result['Contents'] if i["Key"].endswith('.tif')]
    return exists
```

Find data location

[4]:
```
import boto3

path = "/local/path/to/files/" # "Path to directory with rasters:", Right-click on the␣
↪directory and select option to Copy Path

s3 = boto3.client('s3')
file_name = path.split('/', 1)[-1]
if 'shared-buckets' in path:
    file_path = f'shared/{file_name}'
if 'my-private-bucket' in path:
    file_path = f'{user}/{file_name}'
if 'my-public-bucket' in path:
    file_path = f'shared/{user}/{file_name}'
```

(continues on next page)

(continued from previous page)

```
# If using your files
#if file_path:
#    paths = checkFilePath(file_path)
#    files = [ i['Key'] for i in paths ]

# Our sample files
files = ['s3://maap-ops-workspace/shared/alexdevseed/landsat8/viz/Landsat8_30542_comp_
↪cog_2015-2020_dps.tif', 's3://maap-ops-workspace/shared/alexdevseed/landsat8/viz/
↪Landsat8_30543_comp_cog_2015-2020_dps.tif']

print(files)
```

```
['s3://maap-ops-workspace/shared/alexdevseed/landsat8/viz/Landsat8_30542_comp_cog_2015-
↪2020_dps.tif', 's3://maap-ops-workspace/shared/alexdevseed/landsat8/viz/Landsat8_30543_
↪comp_cog_2015-2020_dps.tif']
```

Get raster info (bounds, zoom, data type) from first file in array of files

[5]:
```
import httpx

url = files[0]

r = httpx.get(
    f"{titiler_url}/cog/info",
    params = {
        "url": url,
    }
).json()

if r.get('count') > 0:
    bounds = r.get("bounds")
    min_zoom = r.get("minzoom")
    max_zoom = r.get("maxzoom")
    zoom = min_zoom + 1 if min_zoom == 0 else min_zoom
    bands = r.get("band_metadata")

    print("Bounds:", bounds)
    print("Zoom:", zoom, "min =", min_zoom, "max =", max_zoom)
    print("Data type:", r.get("dtype"))
    print("Bands:", bands)
```

```
Bounds: [-117.10749852280769, 50.78795362739066, -116.50936927974429, 51.16389512140189]
Zoom: 9 min = 9 max = 12
Data type: float32
Bands: [['b1', {}], ['b2', {}], ['b3', {}], ['b4', {}], ['b5', {}], ['b6', {}], ['b7', {}
↪], ['b8', {}], ['b9', {}], ['b10', {}], ['b11', {}], ['b12', {}], ['b13', {}], ['b14',
↪{}], ['b15', {}], ['b16', {}], ['b17', {}], ['b18', {}]]
```

Create mosaic from file urls

[6]:
```
from cogeo_mosaic.mosaic import MosaicJSON

if min_zoom == 0 and max_zoom == 0:
    print("Warning: missing zoom attributes!")

mosaicdata = MosaicJSON.from_urls(files, minzoom=min_zoom, maxzoom=max_zoom)
print("Mosaic created!")
```

```
Mosaic created!
```

Upload the mosaicjson to the TiTiler and filter for `tilejson` results

[7]:
```
mosaic_links = httpx.post(
    url=f"{titiler_url}/mosaics",
    headers={
        "Content-Type": "application/vnd.titiler.mosaicjson+json",
    },
    json=mosaicdata.model_dump(exclude_none=True),
).json()
# print(mosaic_links)
```

[8]:
```
tilejson_object = list(
    filter(lambda x: x.get("rel") == "tilejson", dict(mosaic_links)["links"])
)
tilejson_url = tilejson_object[0]["href"]
tilejson_url
```

[8]:
```
'https://titiler.maap-project.org/mosaics/af2ee61a-b8ee-4ee9-afcc-98b4f4b13ce3/tilejson.
↪json'
```

Calculate raster center for map placement

[9]:
```
r = httpx.get(tilejson_url).json()
center_data = r.get('center')
center = (center_data[1], center_data[0])
zoom = center_data[2]
```

[10]:
```
from shapely.geometry import box

polygon = box(*bounds)
center = (polygon.centroid.y, polygon.centroid.x)
print("Center:", center)
```

```
Center: (50.97592437439628, -116.80843390127599)
```

Create the TileLayer

[11]:
```
from ipyleaflet import TileLayer

params = {
    "return_mask": "true",
    "rescale": f"{band_min}, {band_max}",
    "bidx": "1",
    "colormap_name": "viridis"
}

r = httpx.get(tilejson_url, params=params).json()

layer_url = r['tiles'][0]
custom_layer = TileLayer(url=layer_url, show_loading=True, transparent=True)
```

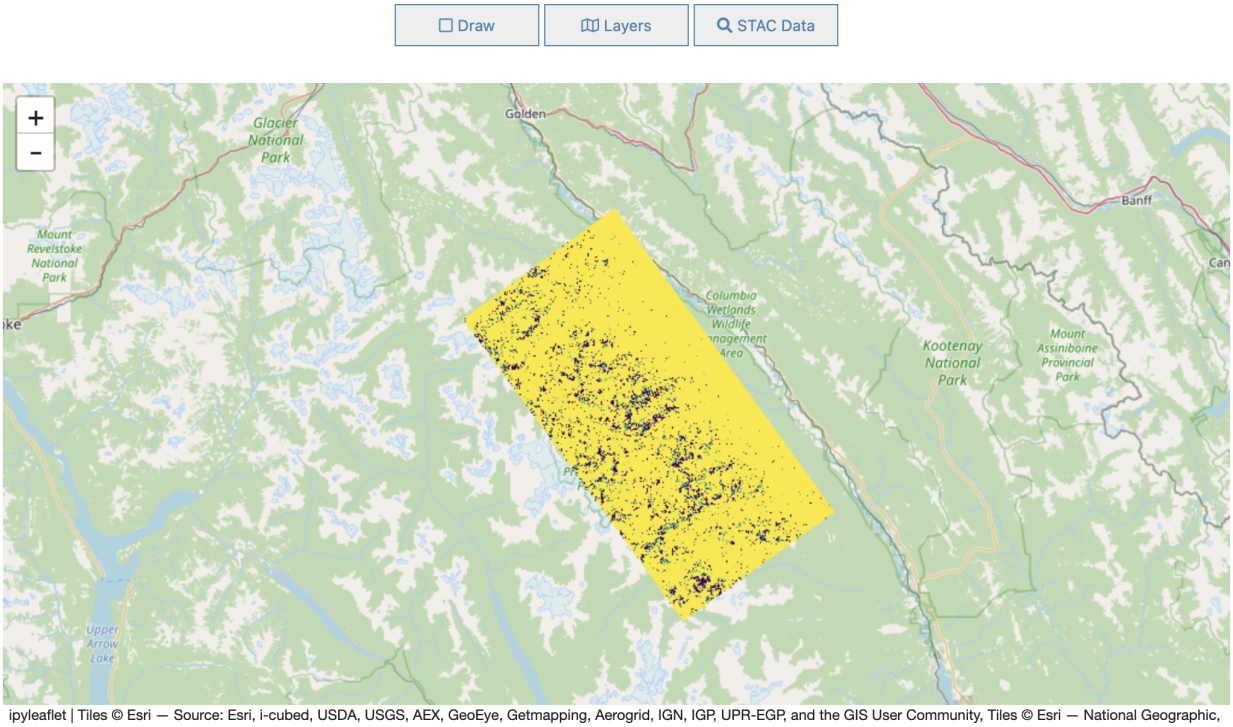
Add the mosaic tile layer to the map

[12]:
```
import stac_ipyleaflet

m = stac_ipyleaflet.StacIpyleaflet(zoom=zoom, center=center)
m.add_layer(custom_layer)
m
```

```
HBox(children=(ToggleButton(value=False, description='Draw', icon='square-o',␣
↪layout=Layout(border_bottom='1px...
```

```
Output()
```

[12]:
```
StacIpyleaflet(center=[50.97592437439628, -116.80843390127599],␣
↪controls=(ZoomControl(options=['position', 'zo...
```



3.2.3 Interval Color Mapping

Author(s): Aimee Barciauskas (DevSeed)

Date: March 3, 2023

Description: In this tutorial, we will pull from a SpatioTemporal Asset Catalog (STAC) collection containing cloud optimized geotiffs (COG) to create a colormap using CSS RGBA values with the COG's data values. We will then use the custom colormap to visualize the Global Aboveground Biomass (AGB) density estimates.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Additional Resources

- Predefined Color Maps
- CSS RGBA Colors
- Using Python Zip Function

Importing and Installing Packages

We will begin by installing any packages we need and importing the packages that we will use.

Prerequisites

- branca
- folium

[ ]:
```
# %pip install folium
# %pip install branca
```

[1]:
```
import branca
from folium import Map, TileLayer
import json
from matplotlib import cm
import requests
```

Discover Data from STAC

[2]:
```
stac_endpoint = "https://stac.maap-project.org"
titiler_endpoint = "https://titiler.maap-project.org"
collection = "NASA_JPL_global_agb_mean_2020"
item = "SAmerica"

items_response = requests.get(f"{stac_endpoint}/collections/{collection}/items/{item}").
↪json()
url = items_response['assets']['mean']['href']
url
```

[2]:
```
's3://nasa-maap-data-store/file-staging/nasa-map/NASA_JPL_global_agb_mean_2020/global_
↪008_06dc_agb_mean_prediction_2020_mosaic_veg_gfccorr_scale1_SAmerica_cog.tif'
```

Get Data Values Using `/statistics` Endpoint

[3]:
```
# You can use gdalinfo /vsis3/nasa-maap-data-store/file-staging/nasa-map/NASA_JPL_global_
↪agb_mean_2020/global_008_06dc_agb_mean_prediction_2020_mosaic_veg_gfccorr_scale1_
↪SAmerica_cog.tif -stats
# or you can get metadata from titiler.
stats_response = requests.get(
    f"{titiler_endpoint}/cog/statistics",
    params = {
        "url": url
```

(continues on next page)

(continued from previous page)

```
    }
).json()
```

[4]:
```
bins = stats_response['b1']['histogram'][1]
bin_ranges = [[bins[i], bins[i+1]] for i in range(len(bins)-1)]
bin_ranges
```

[4]:
```
[[-1166.0, -882.5],
 [-882.5, -599.0],
 [-599.0, -315.5],
 [-315.5, -32.0],
 [-32.0, 251.5],
 [251.5, 535.0],
 [535.0, 818.5],
 [818.5, 1102.0],
 [1102.0, 1385.5],
 [1385.5, 1669.0]]
```

Pick a Color Map and Create a Linear Mapping

[5]:
```
# There are many pre-defined colormaps supported by matplotlib.
# Some are listed below but a complete list be found here: https://matplotlib.org/3.1.0/
↪tutorials/colors/colormaps.html
# You may define custom color maps, but using the predefined ones makes life easier.
sequential_cmaps = [
    'Greys', 'Purples', 'Blues', 'Greens', 'Oranges', 'Reds',
    'YlOrBr', 'YlOrRd', 'OrRd', 'PuRd', 'RdPu', 'BuPu',
    'GnBu', 'PuBu', 'YlGnBu', 'PuBuGn', 'BuGn', 'YlGn'
]

sequential_cmaps2 = [
    'binary', 'gist_yarg', 'gist_gray', 'gray', 'bone', 'pink',
    'spring', 'summer', 'autumn', 'winter', 'cool', 'Wistia',
    'hot', 'afmhot', 'gist_heat', 'copper']
```

[9]:
```
# Here we create a list of pairs, each pair containing a data value interval range (aka
↪"bin")
# with a color value in RGBA (see https://www.w3schools.com/css/css3_colors.asp)
# First we create a list of colors
rgbas = [[int(value) for value in rgb] for rgb in cm.ScalarMappable(cmap='gist_earth').
↪to_rgba(x=bins[:-1], bytes=True)]
# Then we use python's zip function to pair rgba values with data values (https://www.
↪w3schools.com/python/ref_func_zip.asp)
color_map = list(zip(bin_ranges, rgbas))
# some tweaking may be necessary
color_map
```

[9]:
```
[([-1166.0, -882.5], [0, 0, 0, 255]),
 ([-882.5, -599.0], [18, 48, 119, 255]),
 ([-599.0, -315.5], [37, 102, 124, 255]),
 ([-315.5, -32.0], [54, 135, 111, 255]),
```

(continues on next page)

(continued from previous page)

```
 ([-32.0, 251.5], [67, 151, 77, 255]),
 ([251.5, 535.0], [123, 167, 82, 255]),
 ([535.0, 818.5], [169, 179, 91, 255]),
 ([818.5, 1102.0], [191, 164, 100, 255]),
 ([1102.0, 1385.5], [221, 186, 167, 255]),
 ([1385.5, 1669.0], [253, 250, 250, 255])]
```

[7]:
```
# Let's also create a legend using the RGBA values and bins so our map visualization can␣
↪be interpreted!
legend = branca.colormap.StepColormap(rgbas, index=bins, vmin=round(bins[0], 2),␣
↪vmax=round(bins[-1], 2))
```

Preview the data

[10]:
```
# Create a json string of the colormap, so it can be passed as a parameter to titiler's␣
↪API.
cmap = json.dumps(color_map)

# We fetch tilejson from titiler endpoint, to build a better map with appropriate bounds␣
↪and zoom level
tilejson_response = requests.get(
    f"{titiler_endpoint}/cog/tilejson.json",
    params = {
        "url": url,
        "colormap": cmap
    }
).json()

bounds = tilejson_response["bounds"]
m = Map(
    location=((bounds[1] + bounds[3]) / 2,(bounds[0] + bounds[2]) / 2),
    zoom_start=tilejson_response["minzoom"] + 1
)

# We add a TileLayer using the tilejson_response "tiles" value which is the XYZ endpoint␣
↪of titiler.
aod_layer = TileLayer(
    tiles=tilejson_response["tiles"][0],
    opacity=1,
    attr="SAmerica"
)
aod_layer.add_to(m)

# Finally, we add the legend.
legend.caption = 'Global Aboveground Biomass (AGB) Density Estimates in Mg/ha (megagram␣
↪per hectare)'
legend.add_to(m)

m
```

[10]:
```
<folium.folium.Map at 0x10877c890>
```

3.2.4 Visualizing STAC Data Layers using stac_ipyleaflet (beta)

Authors: Emma Paz (Development Seed)

Date: May 10, 2023

Description: This project - **stac_ipyleafet** - leverages the ipyleaflet mapping library and is intended to reduce the code needed to visualize data from the *MAAP STAC*. The beta version supports Cloud-Optimized GeoTIFFs (COGs).

Currently, this package works within this Algorithm Development Environment.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

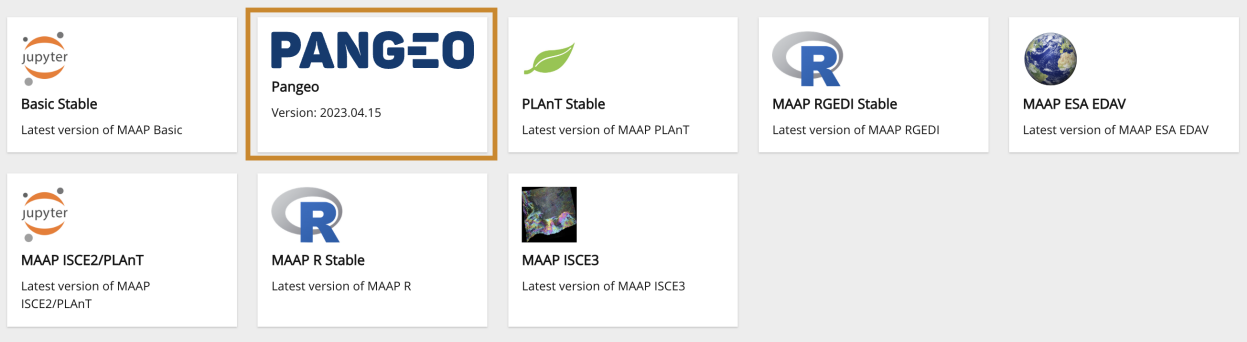
Disclaimer: it is recommended to run this tutorial within MAAP's ADE **Pangeo workspace**, which already includes the stac_ipyleaflet package.

Additional Resources

- stac_ipyleafet repository

Importing and Installing Packages

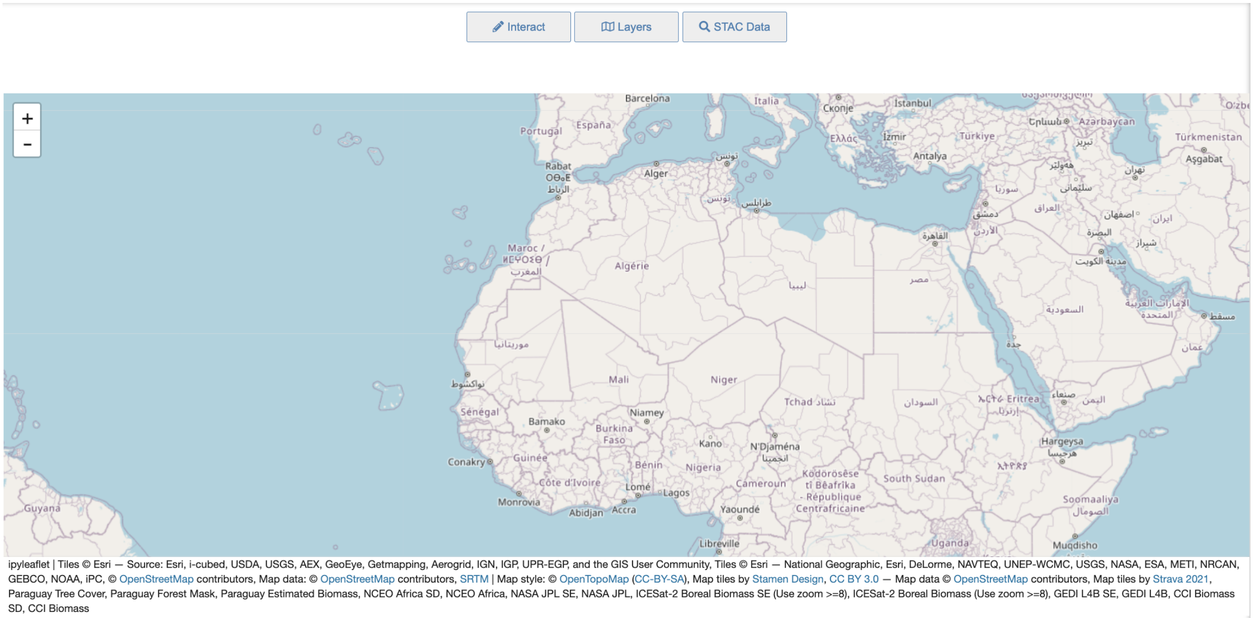
Selecting a new **PANGEO workspace**



or alternatively, you may install stac_ipyleaflet into your environment by uncommenting the install in the following cell:

[1]:
```
# NOTE: This has only been tested in the **MAAP Basic Stable (Vanilla)** environment.

# %pip install git+https://github.com/MAAP-Project/stac_ipyleaflet.git#egg-info=stac_
↪ipyleaflet
```

[ ]:
```
import stac_ipyleaflet
m = stac_ipyleaflet.StacIpyleaflet()
m
```



The **stac ipyleaflet** notebook's user interface consists of a map and a custom set of tools to aid in the discovery and visualization of STAC datasets, along with Biomass Layers and pre-determined Basemaps.

Map Navigation

Move the Map

- press and hold a mouse-click, then drag the map

Adjust the Map's Scale (Zoom Extent) - 1 of 4 Methods:

- click the Zoom In / Out buttons in the top left-corner (this will maintain the center)
- use your mouse's scroll-wheel - hovering over an area of interest
- double-click within the map on an area of interest
- while pressing the `shift` key on your keyboard, press and hold a mouse-click, then drag to draw a rectangle around the area of interest

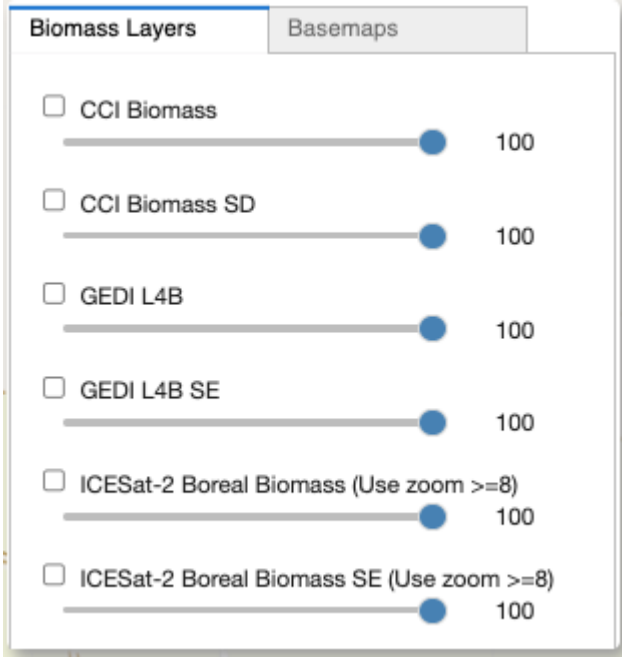
---

Layers Tool

Pressing the Layers button at the top opens the **Layers widget** that consists of 2 tabs. This tool currently allows users to: - View one or more Biomass Layers at the same time to see different combinations. - Choose between common **Basemap Layers** that are known favorites. - Have full control over the **opacity** of any layer or basemap for fine-tuning how the map looks.

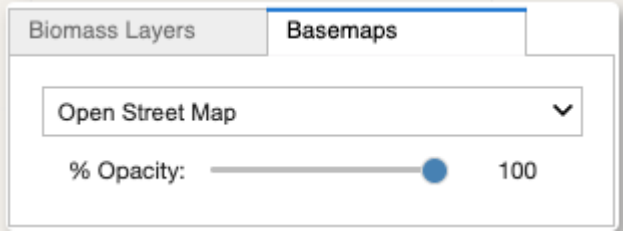
The Layers Tab

- Toggle each layer's visibility by using its checkbox
- Adjust each layer's opacity by moving its slider



The Basemaps Tab

- Select a basemap from the dropdown
- Adjust the basemap's opacity by moving its slider

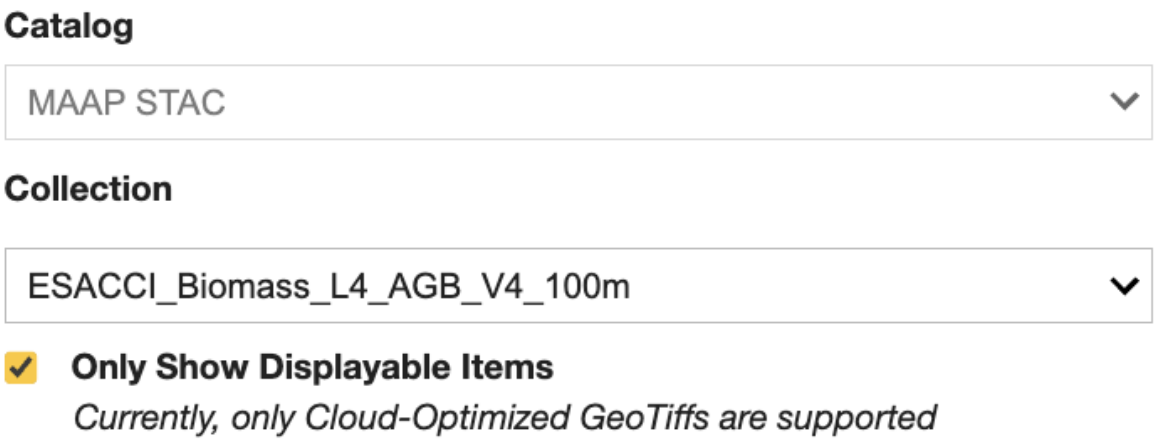


STAC Discovery Tool

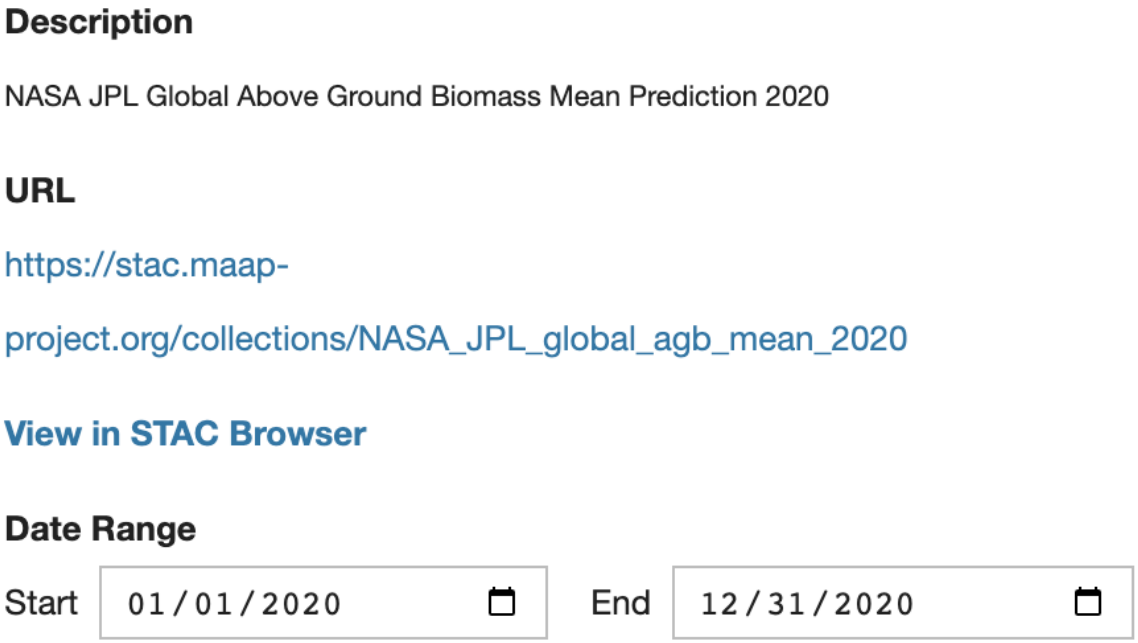
Pressing the STAC Data button at the top opens the **STAC widget** that consists of 2 tabs. This tool currently allows users to: - **Connect** to the MAAP STAC to access collections of mission data. - **Discover** items per the selected collection, including description, available dates, & direct URL. - **Identify** valid COG datasets. - **Add COG tiles** dynamically to the map. - **Customize** the tiles by changing the selected color palette for the selected item.

The Catalog Data Discovery Tab

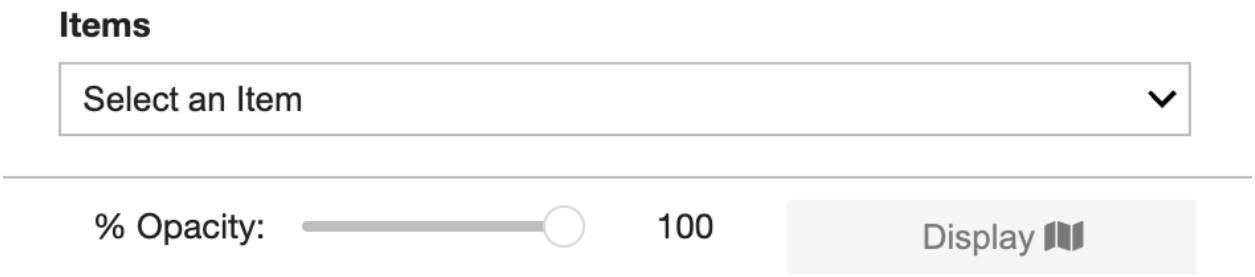
- Select a Collection within the default STAC library.



- Browse through the Collection's details.

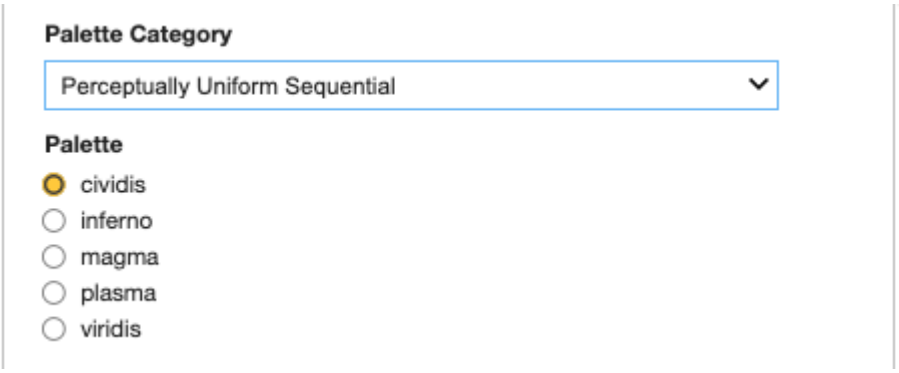


- Select an item from the collection to check if it is a valid COG. If it is, the Display button will become active (available) to add the selected item to the map. The displayed STAC layer's opacity can be adjusted by moving its slider.



The Visualization Tab

- Select a category from the dropdown.
- Select an item from the corresponding color palettes.
- Press the Display button to update the data on the map.



Interact with the Map

1. **Activate the Interact Tools** (click on the top *Interact* button)
2. From within the `Point` tab: **Use your mouse** to activate the Point tool; then click on the map at a location of interest
    - **Coordinates** will be printed within the open tab
    - **Raster cell values** will be identified and printed, if raster layers are on
3. From within the `Area` tab: **Use your mouse** to activate the Polygon tool; then click, hold and draw a polygon over the map - releasing to finish
    - The area of interest's **Coordinates & BBox** within the open tab

- Alternatively **Print** the area of interest's bbox from within a cell:

```
[3]: m.aoi_bbox

[3]: (-18.354806, 21.524308, -10.794943, 28.603513)
```

4. **Clear** the point or polygon graphics as needed

Add a Tile Layer

```
[3]: import httpx
import json
titiler_url = "https://titiler.maap-project.org"  # MAAP titiler endpoint
titiler_tilejson_url = f"{titiler_url}/cog/tilejson.json"
band_min = 0
band_max = 400
bidx = 1
colormap_name = 'gist_earth_r'
data_url = f"s3://maap-ops-workspace/shared/alexdevseed/landsat8/viz/Copernicus_30439_
↪covars_cog_topo_stack.tif"
```

```
[8]: from ipyleaflet import TileLayer

params = {
    "url": data_url,
    "bidx": bidx,
    "rescale": f"{band_min}, {band_max}",
    "pixel_selection": "first",
    "resampling_method": "nearest",
    "colormap_name": colormap_name
}
r = httpx.get(titiler_tilejson_url, params = params).json()

custom_layer = TileLayer(url=r["tiles"][0], show_loading=True, visible=True)
# print(custom_layer)
```

```
TileLayer(options=['attribution', 'bounds', 'detect_retina', 'max_native_zoom', 'max_zoom
↪', 'min_native_zoom', 'min_zoom', 'no_wrap', 'tile_size', 'tms', 'zoom_offset'], show_
↪loading=True, url='https://titiler.maap-project.org/cog/tiles/WebMercatorQuad/{z}/{x}/
↪{y}@1x?url=s3%3A%2F%2Fmaap-ops-workspace%2Fshared%2Falexdevseed%2Flandsat8%2Fviz
↪%2FCopernicus_30439_covars_cog_topo_stack.tif&bidx=1&rescale=0%2C+400&pixel_
↪selection=first&resampling_method=nearest&colormap_name=gist_earth_r')
```

```
[9]: r = httpx.get(
    f"{titiler_url}/cog/info",
    params = {
        "url": data_url,
    }
).json()
```

(continues on next page)

(continued from previous page)

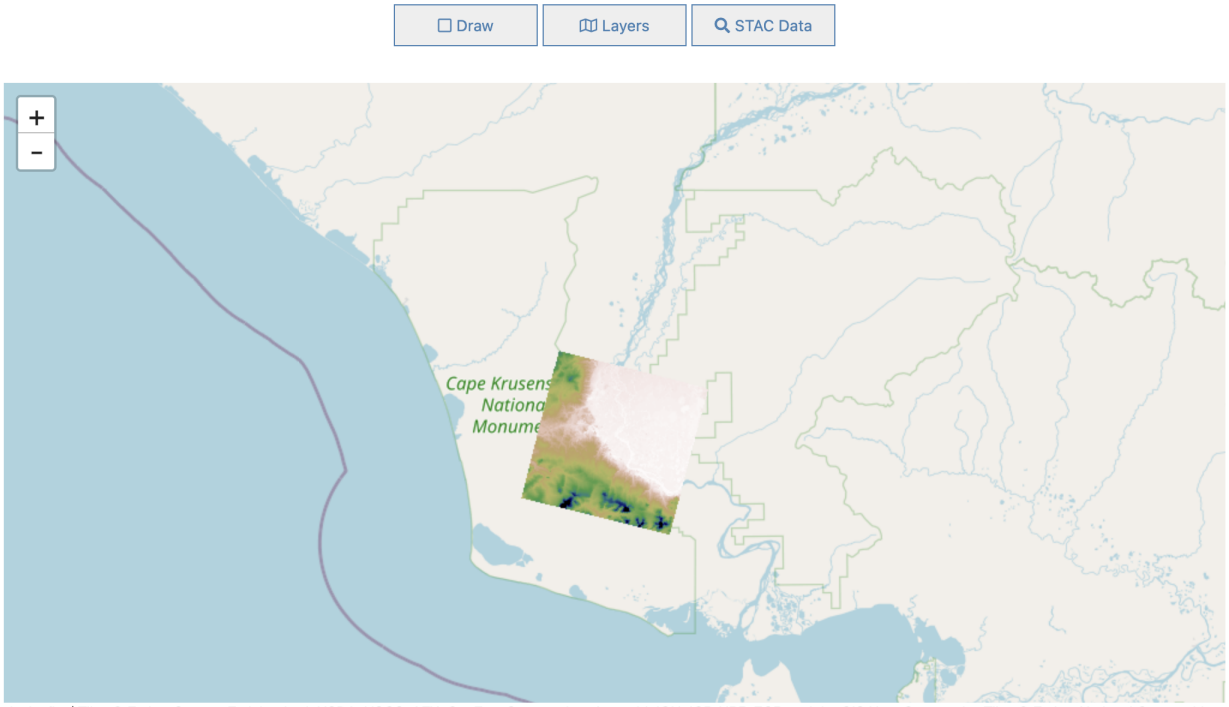
```
# print(json.dumps(r, indent=4))

bounds = r.get("bounds")
minzoom = r.get("minzoom")
zoom = minzoom + 1 if minzoom == 0 else minzoom
bands = r.get("band_metadata")

print("Bounds:", bounds)
print("Zoom:", zoom)
print("Data type:", r.get("dtype"))
print("Bands:", bands)
```

```
Bounds: [-163.51338693693168, 67.17121197506852, -162.6566451826878, 67.49580310072406]
Zoom: 8
Data type: float32
Bands: [['b1', {}], ['b2', {}], ['b3', {}], ['b4', {}], ['b5', {}]]
```

[13]:
```
from shapely.geometry import box

polygon = box(*bounds)
center = (polygon.centroid.y, polygon.centroid.x)
print("Center:", center)
```

```
Center: (67.33350753789628, -163.0850160598097)
```

[ ]:
```
# import stac_ipyleaflet

m = stac_ipyleaflet.StacIpyleaflet(zoom=zoom, center=center)
m.add_layer(custom_layer)
m
```




```
m.remove_layer(custom_layer)
```

3.3 Access

3.3.1 Accessing Data from the MAAP

Authors: Samuel Ayers (UAH), Brian Satorius (JPL)

Date: May 26, 2022 (Revised July 2023)

Description: In this example, we demonstrate how to access data from the MAAP using the 'getData' method of the maap-py library. At this time, this procedure is the same for user-contributed data added to the store.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Additional Resources

- Search Granules Tutorial
- Additional Search Attributes

Importing Packages

We import the `os` module, import the `MAAP` package, and create a new MAAP class.

[1]:
```
# import os module
import os

# import the MAAP package
from maap.maap import MAAP

# create MAAP class
maap = MAAP()
```

For this example, the additional bounding_box attribute is used to search for granules within the Mondah Forest Gabon research site spatial area. For more information about searching for granules in MAAP, please see https://docs.maap-project.org/en/latest/technical_tutorials/search/granules.html.

[2]:
```
SHORTNAME = "AFLVIS2"
BBOX = '9.316216,0.538705,9.422509,0.616939'

# search for granules with SHORTNAME
results = maap.searchGranule(
    short_name=SHORTNAME,
    bounding_box=BBOX
)
results[0]
```

[2]:
```
{'concept-id': 'G1549416185-NSIDC_ECS',
 'collection-concept-id': 'C1549378743-NSIDC_ECS',
 'revision-id': '2',
 'format': 'application/echo10+xml',
 'Granule': {'GranuleUR': 'SC:AFLVIS2.001:138348928',
  'InsertTime': '2018-09-24T11:01:33.892Z',
  'LastUpdate': '2018-09-24T11:02:09.869Z',
  'Collection': {'DataSetId': 'AfriSAR LVIS L2 Geolocated Surface Elevation Product V001
↪'},
  'DataGranule': {'SizeMBDataGranule': '130.082',
   'ProducerGranuleId': 'LVIS2_Gabon2016_0304_R1808_054746.TXT',
   'DayNightFlag': 'UNSPECIFIED',
   'ProductionDateTime': '2018-08-23T18:32:04.000Z',
   'LocalVersionId': '001'},
  'Temporal': {'RangeDateTime': {'BeginningDateTime': '2016-03-04T15:12:26.391000Z',
    'EndingDateTime': '2016-03-04T15:19:18.916000Z'}},
  'Spatial': {'HorizontalSpatialDomain': {'Geometry': {'GPolygon': {'Boundary': {'Point':
↪ [{'PointLongitude': '9.27281',
        'PointLatitude': '0.57138'},
       {'PointLongitude': '9.33743', 'PointLatitude': '0.57138'},
       {'PointLongitude': '9.42716', 'PointLatitude': '0.57138'},
```

(continues on next page)

(continued from previous page)

```
        {'PointLongitude': '9.5169', 'PointLatitude': '0.57138'},
        {'PointLongitude': '9.60664', 'PointLatitude': '0.57138'},
        {'PointLongitude': '9.66048', 'PointLatitude': '0.56777'},
        {'PointLongitude': '9.73945', 'PointLatitude': '0.56416'},
        {'PointLongitude': '9.73586', 'PointLatitude': '0.54613'},
        {'PointLongitude': '9.64972', 'PointLatitude': '0.54974'},
        {'PointLongitude': '9.55998', 'PointLatitude': '0.54974'},
        {'PointLongitude': '9.47024', 'PointLatitude': '0.54974'},
        {'PointLongitude': '9.3805', 'PointLatitude': '0.54974'},
        {'PointLongitude': '9.32307', 'PointLatitude': '0.54613'},
        {'PointLongitude': '9.2764', 'PointLatitude': '0.54974'},
        {'PointLongitude': '9.27281', 'PointLatitude': '0.56416'},
        {'PointLongitude': '9.27281', 'PointLatitude': '0.56777'}]}}}}},
  'Platforms': {'Platform': {'ShortName': 'B-200',
    'Instruments': {'Instrument': {'ShortName': 'LVIS',
      'Sensors': {'Sensor': {'ShortName': 'LVIS'}}}}}},
  'Campaigns': {'Campaign': {'ShortName': 'AfriSAR'}},
  'AdditionalAttributes': {'AdditionalAttribute': [{'Name': 'SIPSMetGenVersion',
     'Values': {'Value': '2.5'}},
    {'Name': 'ThemeID', 'Values': {'Value': 'AfriSAR'}},
    {'Name': 'AircraftID', 'Values': {'Value': 'N529NA'}}]},
  'OnlineAccessURLs': {'OnlineAccessURL': [{'URL': 'https://n5eil01u.ecs.nsidc.org/DP8/
↪ICEBRIDGE/AFLVIS2.001/2016.03.04/LVIS2_Gabon2016_0304_R1808_054746.TXT',
     'MimeType': 'application/octet-stream'}]},
  'OnlineResources': {'OnlineResource': {'URL': 'https://n5eil01u.ecs.nsidc.org/DP8/
↪ICEBRIDGE/AFLVIS2.001/2016.03.04/LVIS2_Gabon2016_0304_R1808_054746.TXT.xml',
    'Type': 'METADATA',
    'MimeType': 'text/xml'}},
  'Orderable': 'true',
  'Visible': 'true'}}
```

We assign a variable (in this case, `data_file`) to the first result of our search from the cell above.

A data directory is then set, and if the directory does not already exist, it is created. The file from our search is then downloaded into the file system in this directory. Here, the function getData() is downloading the data using the access URLs in the CMR granule and downloading it directly to the path provided.

[3]:
```
# grab first result
data_file = results[0]

# set data directory
dataDir = './data'

# check if directory exists -> if directory doesn't exist, directory is created
if not os.path.exists(dataDir):
    os.mkdir(dataDir)

# extract the link to the resource
data = data_file.getData(dataDir)
```

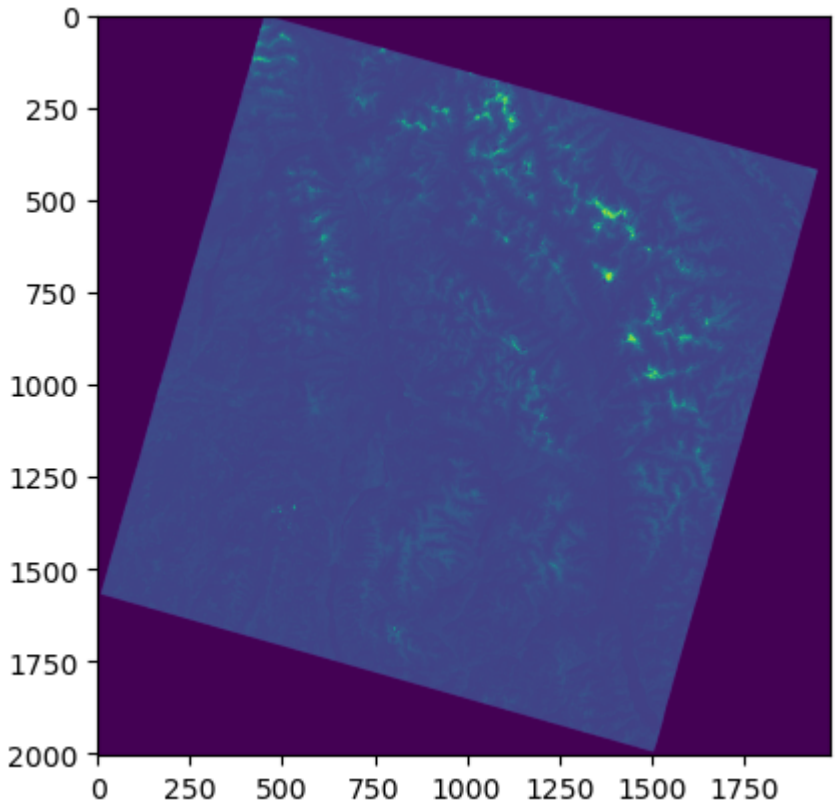
We can now see that the data directory has been created and the data file is downloaded into the directory. The downloaded file remains in the data directory until the user deletes it.

Accessing Data from AWS Requester Pays Buckets

Some data is cloud available but in requester pays buckets. In this example, we use Rasterio, Boto3, and MAAP's `aws.requester_pays_credentials()` function to retrieve data within the `usgs-landsat` requester pays bucket.

[4]:
```
import boto3
import rasterio as rio

from maap.maap import MAAP
from rasterio.plot import show
from rasterio.session import AWSSession


maap = MAAP(maap_host='api.maap-project.org')
credentials = maap.aws.requester_pays_credentials()

boto3_session = boto3.Session(
    aws_access_key_id=credentials['aws_access_key_id'],
    aws_secret_access_key=credentials['aws_secret_access_key'],
    aws_session_token=credentials['aws_session_token']
)
```

[5]:
```
aws_session = AWSSession(boto3_session, requester_pays=True)
file_s3 = 's3://usgs-landsat/collection02/level-2/standard/oli-tirs/2015/044/025/LC08_
↪L2SP_044025_20150812_20200908_02_T1/LC08_L2SP_044025_20150812_20200908_02_T1_SR_B2.TIF'
with rio.Env(aws_session):
    with rio.open(file_s3, 'r') as src:
        # list of overviews
        oviews = src.overviews(1)
        # get second item from list to retrieve a thumbnail
        oview = oviews[1]
        # read first band of file and set shape of new output array
        thumbnail = src.read(1, out_shape=(1, int(src.height // oview), int(src.width //␣
↪oview)))
# now display the thumbnail
show(thumbnail)
```



[5]:
```
<Axes: >
```

You may adjust the expiration time of the AWS credentials generated by `maap.aws.requester_pays_credentials()`:

[ ]:
```
# Credential expiration time in seconds (defaults to 12 hours)
maap.aws.requester_pays_credentials(expiration=3600)
```

3.3.2 Accessing Data Provided by NASA's Distributed Active Archive Centers (DAACs)

Authors: Samuel Ayers (UAH), Alex Mandel (DevSeed)

Date: April 3, 2023

Description: In this example, we will demonstrate how to get authorized access from a Distributed Active Archive Center (DAAC) in NASA Earthdata, and then access data from the DAAC. Note: not all DAACs will require prior authorization to access data.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

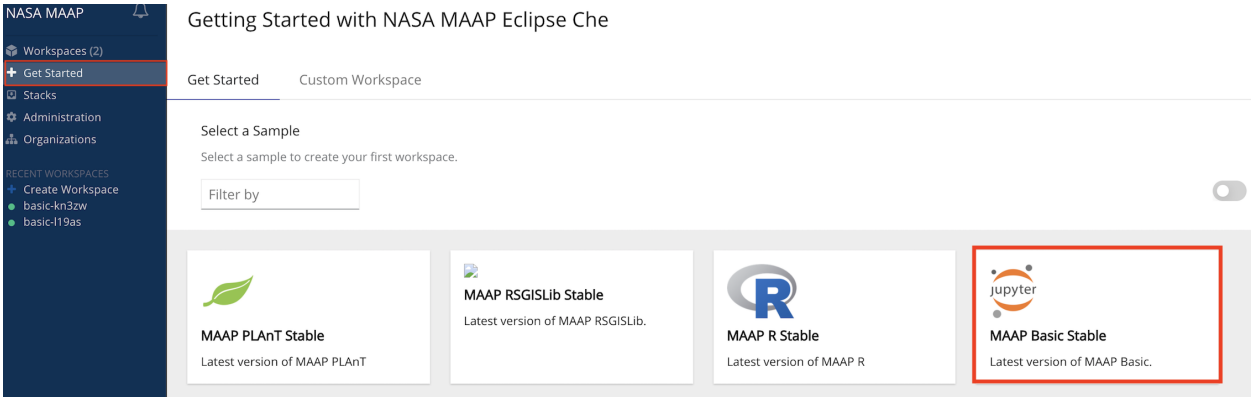
Additional Resources

- Changes to Land Processes Distributed Active Archive Center (LP DAAC) Data Access
- LP DAAC AppEEARS API Documentation
- Alaska Satellite Facility (ASF) API Basics

Create a Workspace

It is possible to download data provided by DAACs, including data which is not cataloged by the MAAP's CMR, using the NASA MAAP ADE. This data is hosted externally from the MAAP but can be accessed using the NASA MAAP ADE's authentication systems.

In order to do this, we start by creating a Jupyter workspace within the NASA MAAP ADE. Using the left-hand navigation, select "+ Get Started" and then select the "Jupyter - MAAP Basic Stable" workspace.



Alternatively, you can create a workspace using the "Workspaces" interface. See Create Workspace for more information.

Importing Packages

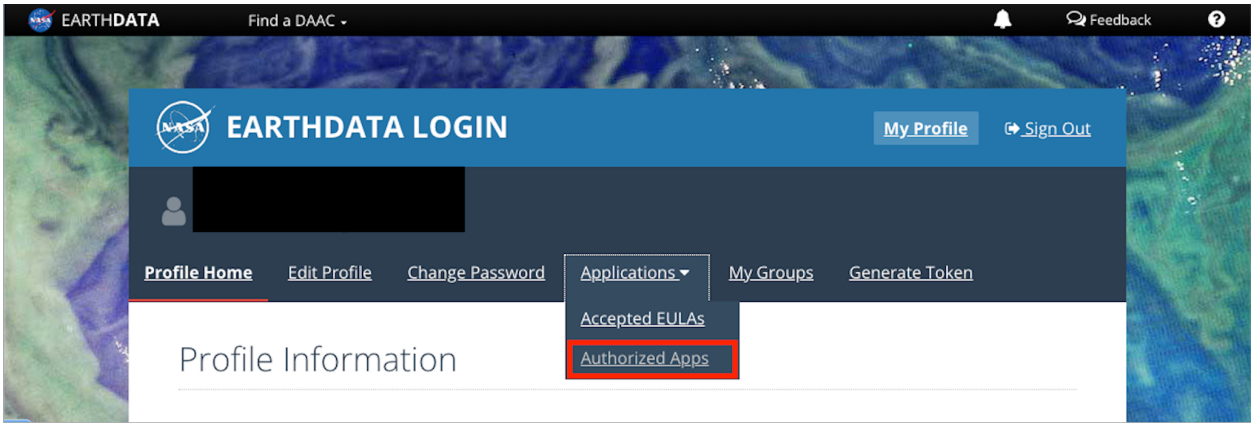
You can access data from Jupyter Notebook within your workspace. In your Jupyter Notebook, start by importing the `maap` package. Then invoke the `MAAP` constructor, setting the `maap_host` argument to `'api.maap-project.org'`.

[7]:
```
# import the maap package
from maap.maap import MAAP
#import os for directory creation later in the example
import os
# invoke the MAAP constructor using the maap_host argument
maap = MAAP(maap_host='api.maap-project.org')
```

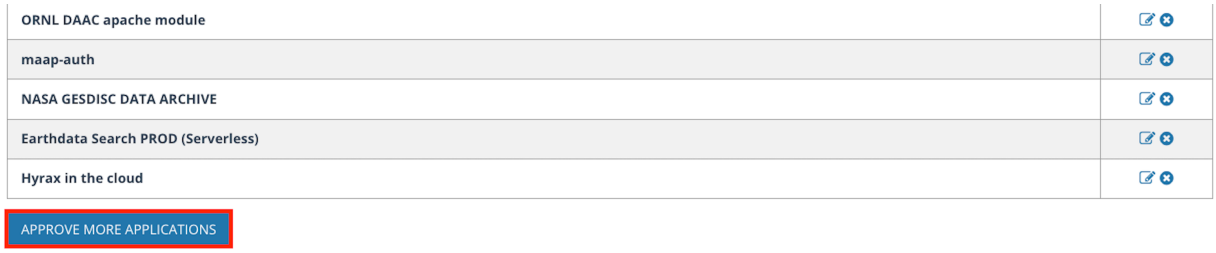
Granting Earthdata Login Access

In order to access external DAAC data from the NASA MAAP ADE, MAAP uses your Earthdata Login profile to send a data request to the desired DAAC application.

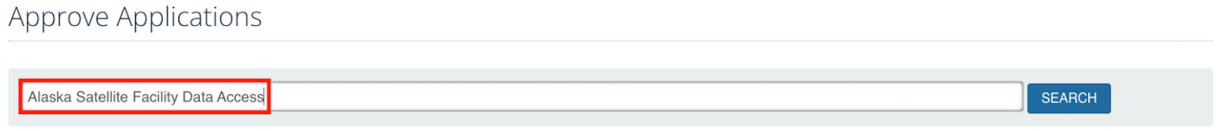
Some DAAC applications (such as 'Alaska Satellite Facility Data Access') must be authorized before you can use them. Login or register at https://urs.earthdata.nasa.gov/ in order to see the applications that you have authorized. From the profile page, click on the 'Applications' tab and select 'Authorized Apps' from the drop-down menu.



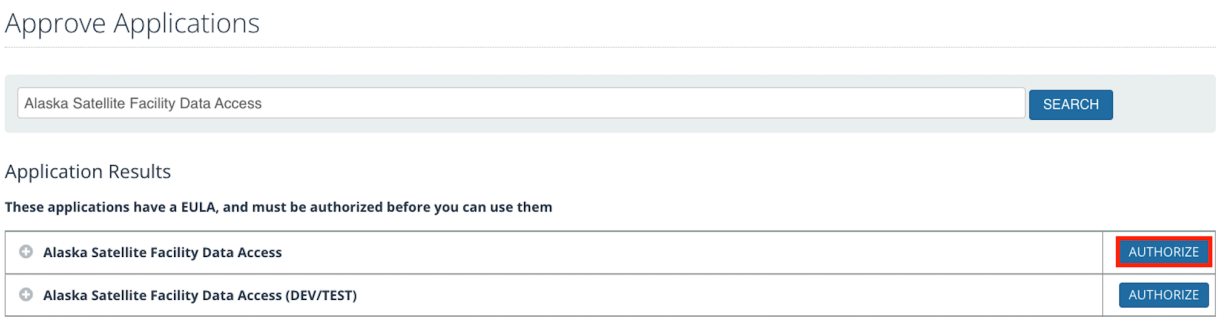
This takes you to the Approved Applications page which lists the applications you have authorized. To add more applications, scroll down to the bottom of the page and click the 'APPROVE MORE APPLICATIONS' button which takes you to the Application search page.



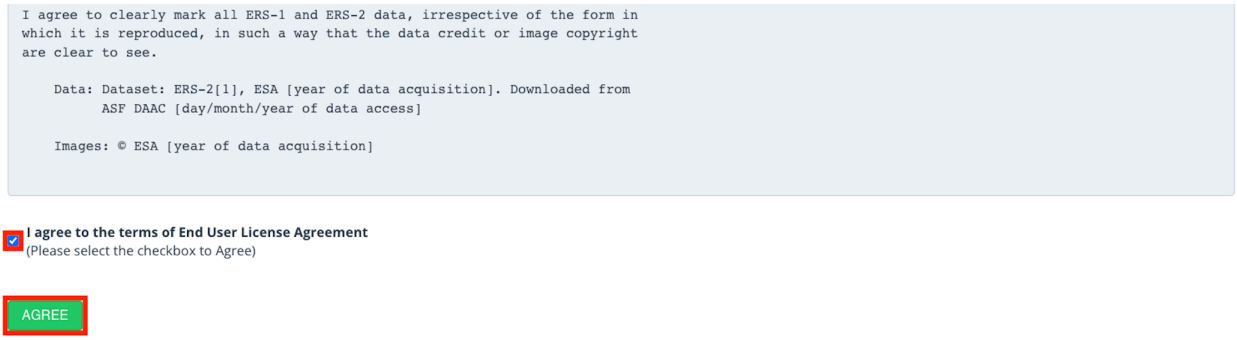
Enter the desired application name within the search box and click the 'SEARCH' button. After this, a list of search results appears.



Once you find the desired application, click the 'AUTHORIZE' button next to the name.



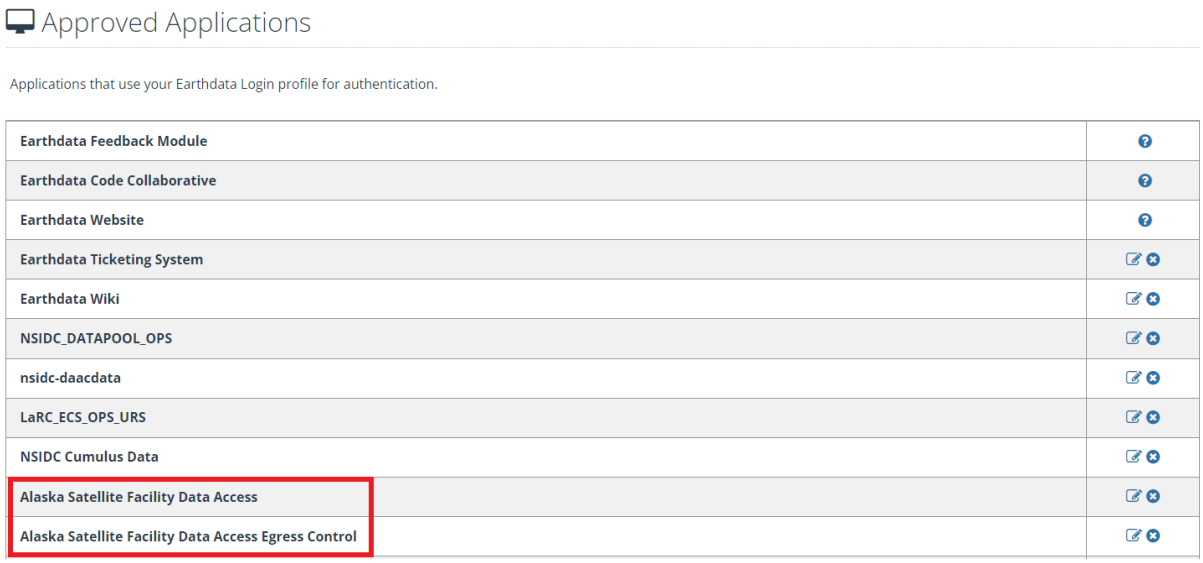
You are then presented with its End User License Agreement. In order to have authorization, you need to select the 'I agree to the terms of End User License Agreement' checkbox and then click the 'AGREE' button.



Along with 'Alaska Satellite Facility Data Access', you must also authorize 'Alaska Satellite Facility Data Access Egress Control'. Return to the search page to find this application.

Similar to the previous application, once you click 'AUTHORIZE', you will also select the 'I agree to the terms of the End User License Agreement' checkbox and click 'AGREE'.

After these steps are done, you are then shown the Approved Applications page again and both of the desired applications should now be listed.



Note that if Earthdata Login access is not granted to your target DAAC application, the following example will result

in a 401-permission error.

Accessing Sentinel-1 Granule Data from the Alaska Satellite Facility (ASF)

Search for a granule using the `searchGranule` function (for more information on searching for granules, see Searching for Granules in MAAP). Then utilize the `getData` function, which downloads granule data if it doesn't already exist locally. We can use `getData` to download the first result from our granule search into the file system and assign it to a variable (in this case `download`). Note that you will need to authorize the 'Alaska Satellite Facility Data Access' application before downloading any results from our search (see the above section for more information concerning authorizing applications).

Before downloading our data, we will also create a new data directory to download our files into using `os`.

[8]:
```
# set data directory
dataDir = './data'

# check if directory exists -> if directory doesn't exist, directory is created
if not os.path.exists(dataDir):
    os.mkdir(dataDir)
```

[9]:
```
# search for granule data using the short_name argument
results = maap.searchGranule(short_name='SENTINEL-1A_DP_GRD_HIGH')
# download first result
download = results[0].getData(dataDir)
```

Note that we can then use the `print` function to see the file name and directory.

[10]:
```
# print file directory
print(download)
```

```
./data/S1A_S3_GRDH_1SDH_20140615T034444_20140615T034512_001055_00107C_8977.zip
```

Accessing Harmonized Landsat Sentinel-2 (HLS) Level 3 Granule Data from the Land Processes Distributed Active Archive Center (LP DAAC)

We use a similar approach in order to access HLS Level 3 granule data. Note that this data is not cataloged by the MAAP's CMR but we can use `searchGranule`'s `cmr_host` argument to specify a CMR instance external to MAAP.

[11]:
```
# search for granule data using CMR host name and short name arguments
results = maap.searchGranule(
    cmr_host='cmr.earthdata.nasa.gov',
    short_name='HLSL30')
# download first result
download = results[0].getData(dataDir)
```

As in the previous example, we can use the `print` function to see the file name and directory.

[12]:
```
# print file directory
print(download)
```

```
./data/HLS.L30.T59WPT.2013101T001445.v2.0.B09.tif
```

3.3.3 Accessing Cloud Optimized Data

Authors: Samuel Ayers (UAH)

Date: April 28, 2021 (Revised July 2023)

Description: The following is an example that uses Shuttle Radar Topography Mission (SRTM) Cloud Optimized GeoTIFF (COG) data from the MAAP data store, via MAAP STAC search. In this example, we read in elevation data using a bounding box tile.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Additional Resources

- Using pystac-client

Importing and Installing Packages

To be able to run this notebook you'll need the following requirements:

- rasterio
- folium
- geopandas
- rio-cogeo

If the packages below are not installed already, uncomment the following cell

[1]:
```
# %pip install -U folium geopandas rasterio>=1.2.3 rio-cogeo
# %pip install pystac-client
```

[2]:
```
# import the maap package to handle queries
from maap.maap import MAAP

# invoke the MAAP
maap = MAAP()
```

Retrieving the Data

We can use pystac_client `get_collection` to retrieve the desired collection, in this case the SRTMGL1_COD collection and set the result of the function to a variable.

[3]:
```
# Get the SRTMGL1_COD collection
from pystac_client import Client
catalog = 'https://stac.maap-project.org/'
client = Client.open(catalog)
```

(continues on next page)

(continued from previous page)

```
collection = client.get_collection('SRTMGL1_COD')
collection
```

[3]:
```
<CollectionClient id=SRTMGL1_COD>
```

We can narrow the area of interest for the items within the collection by passing bounding box values into the pystac_client `search` function.

[4]:
```
# set bounding box
bbox = "-101.5,45.5,-100.5,46.5"

# retreive STAC items from the collection that are within the bounding box
search = client.search(bbox=bbox, collections=collection, max_items=20)
```

Let's check how many items we are working with.

[5]:
```
# show number of items in search results
len(list(search.items()))
```

[5]:
```
4
```

Inspecting the Results

Now we can work on inspecting our results. In order to do this, we import the `geopandas`,`shapely`, and `folium` packages.

[6]:
```
# import geopandas to work with geospatial data
import geopandas as gpd

# import shapely for manipulation and analysis of geometric objects
from shapely.geometry import shape, box

# import folium to visualize data in an interactive leaflet map
import folium
```

We can use the `gpd.GeoSeries` function to create a GeoSeries of all the shapely polygons created from the geometries of our item results. According to the GeoPandas documentation, a GeoSeries is a "Series [a type of one-dimensional array] object designed to store shapely geometry objects". We use 'EPSG:4326' (WGS 84) for the coordinate reference system then we can check the GeoSeries.

[7]:
```
# create GeoSeries of all polygons from granule results with WGS 84 coordinate reference␣
↪system
geometries = gpd.GeoSeries(
    [shape(item.geometry) for item in search.items()],
    crs='EPSG:4326'
)

# check GeoSeries
geometries
```

[7]:
```
0    POLYGON ((-102.00014 45.99986, -100.99986 45.9...
1    POLYGON ((-101.00014 45.99986, -99.99986 45.99...
```

(continues on next page)

(continued from previous page)

```
2    POLYGON ((-102.00014 44.99986, -100.99986 44.9...
3    POLYGON ((-101.00014 44.99986, -99.99986 44.99...
dtype: geometry
```

Now we create a list from our bounding box values. Then we use the `centroid` function to get the centroid of our bounding box and set to a point. Next we use `folium.Map` to create a map. For the map's parameters, let's set the centroid point coordinates as the location, "cartodbpositron" for the map tileset, and 7 as the starting zoom level. With our map created, we can create a dictionary containing style information for our bounding box. Then we can use `folium.GeoJson` to create GeoJsons from our `geometries` GeoSeries and add them to the map. We also use the `folium.GeoJson` function to create a GeoJson of a polygon created from our bounding box list, name it, and add our style information. Finally, we check the map by displaying it.

[8]:
```
# create list of bounding box values
bbox_list =  [float(value) for value in bbox.split(',')]

# get centroid point from bounding box values
center = box(*bbox_list).centroid

# create map with folium with arguments for lat/lon of map, map tileset, and starting␣
↪zoom level
m = folium.Map(
    location=[center.y,center.x],
    tiles="cartodbpositron",
    zoom_start=7,
)
# create style information for bounding box
bbox_style = {'fillColor': '#ff0000', 'color': '#ff0000'}

# create GeoJson of `geometries` and add to map
folium.GeoJson(geometries, name="tiles").add_to(m)

# create GeoJson of `bbox_list` polygon and add to map with specified style
folium.GeoJson(
    box(*bbox_list),
    name="bbox",
    style_function=lambda x:bbox_style
).add_to(m)

# display map
m
```

[8]:
```
<folium.folium.Map at 0x7f50e9d3b5d0>
```

Creating a Mosaic

Let's check the raster information contained in our item results. In order to do this, we import some more packages. - To read and write files in raster format, import `rasterio`. - From `rasterio` we import `merge` to copy valid pixels from an input into an output file - `AWSSession` to set up an Amazon Web Services (AWS) session - `show` to display images and label axes - Import `boto3` in order to work with AWS - From `matplotlib`, we want to import `imshow` which allows us to display images from data - Import `numpy` to work with multi-dimensional arrays and `numpy.ma` to work with masked arrays. - From `pyproj`, import `Proj` for converting between geographic and projected coordinate reference systems and `Transformer` to make transformations.

[9]:
```
# import rasterio for reading and writing in raster format
import rasterio as rio

# copy valid pixels from input files to an output file.
from rasterio.merge import merge

# set up AWS session
from rasterio.session import AWSSession

# display images, label axes
from rasterio.plot import show

# import boto3 to work with Amazon Web Services
import boto3

# display images from data
from matplotlib.pyplot import imshow

# import numpy to work with multi-dimensional arrays
import numpy as np

# import numpy.ma to work with masked arrays
import numpy.ma as ma

# convert between geographic and projected coordinates and make transformations
from pyproj import Proj, Transformer
```

Finally, we import `os` and run some code in order to speed up Geospatial Data Abstraction Library (GDAL) reads from Amazon Simple Storage Service (S3) buckets by skipping sidecar (connected) files.

[10]:
```
# speed up GDAL reads from S3 buckets by skipping sidecar files
import os
os.environ['GDAL_DISABLE_READDIR_ON_OPEN'] = 'EMPTY_DIR'
```

Now that we have the necessary packages, let's get a list of S3 urls to the granules. To start, set up an AWS session. The S3 urls are contained within the `search.items()` assets, so we loop through the items in our results to get the S3 urls and add them to a new list. We can then use the `sort` function to sort the S3 urls in an acending order. Then we can check our S3 url list.

[11]:
```
# set up AWS session
aws_session = AWSSession(boto3.Session())

# get the S3 urls to the granules
file_S3 = [item.assets["cog_default"].href for item in search.items()]
```

(continues on next page)

(continued from previous page)

```
# sort list in ascending order
file_S3.sort()

# check list
file_S3
```

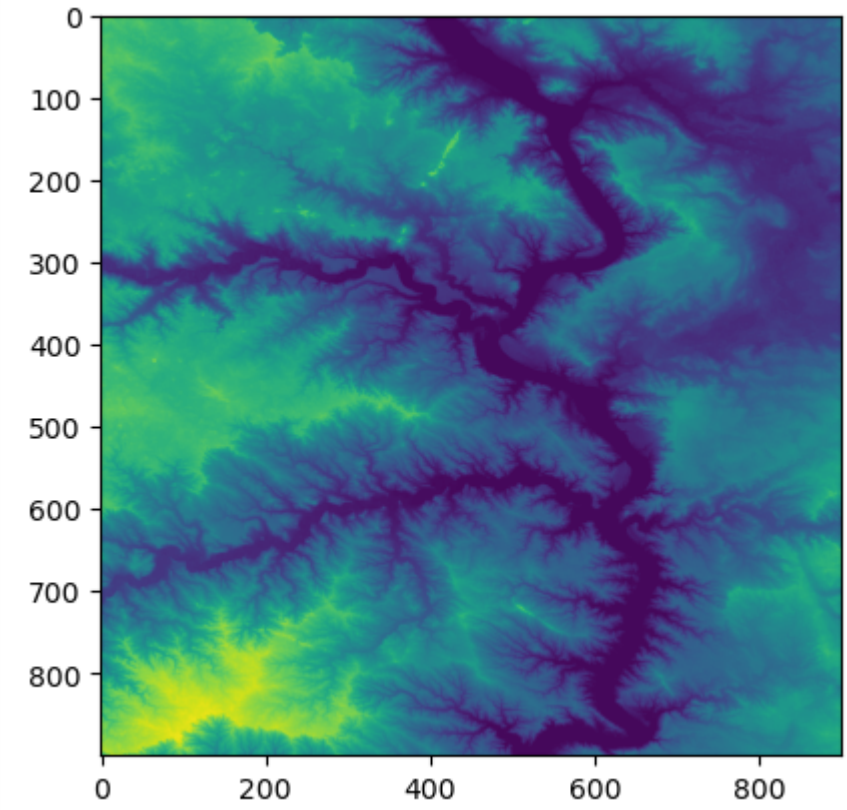
[11]:
```
['s3://nasa-maap-data-store/file-staging/nasa-map/SRTMGL1_COD___001/N45W101.SRTMGL1.tif',
 's3://nasa-maap-data-store/file-staging/nasa-map/SRTMGL1_COD___001/N45W102.SRTMGL1.tif',
 's3://nasa-maap-data-store/file-staging/nasa-map/SRTMGL1_COD___001/N46W101.SRTMGL1.tif',
 's3://nasa-maap-data-store/file-staging/nasa-map/SRTMGL1_COD___001/N46W102.SRTMGL1.tif']
```

We can now check to see that we can read the AWS files and display a thumbnail. Pass the boto3 session to `rio.Env`, which is the GDAL/AWS environment for `rasterio`. Use the `rio.open` function to read in one of the Cloud Optimized GeoTIFFs.

Now let's use the `overviews` command to get a list of overviews. Overviews are versions of the data with lower resolution, and can thus increase performance in applications. Let's get the first overview from our list for retrieving a thumbnail.

Retrieve a thumbnail by reading the first band of our file and setting the shape of the new output array. The shape can be set with a tuple of integers containing the number of datasets as well as the `height` and `width` of the file divided by our integer from the overview list. Now use the `show` function to display the thumbnail.

[12]:
```
# prove that we can read the AWS files
# for more information - https://automating-gis-processes.github.io/CSC/notebooks/L5/
↪read-cogs.html
with rio.Env(aws_session):
    with rio.open(file_S3[0], 'r') as src:
        # list of overviews
        oviews = src.overviews(1)
        # get second item from list to retrieve a thumbnail
        oview = oviews[1]
        # read first band of file and set shape of new output array
        thumbnail = src.read(1, out_shape=(1, int(src.height // oview), int(src.width //␣
↪oview)))
# now display the thumbnail
show(thumbnail)
```



[12]:
```
<Axes: >
```

Since we verified that we can read the AWS files and display a thumbnail, we can create a mosaic from all of the rasters in our `file_S3` list. To do this, again pass the boto3 session to `rio.Env`. Then create a list which contains all of the read in Cloud Optimized GeoTIFFs (this may take a while).

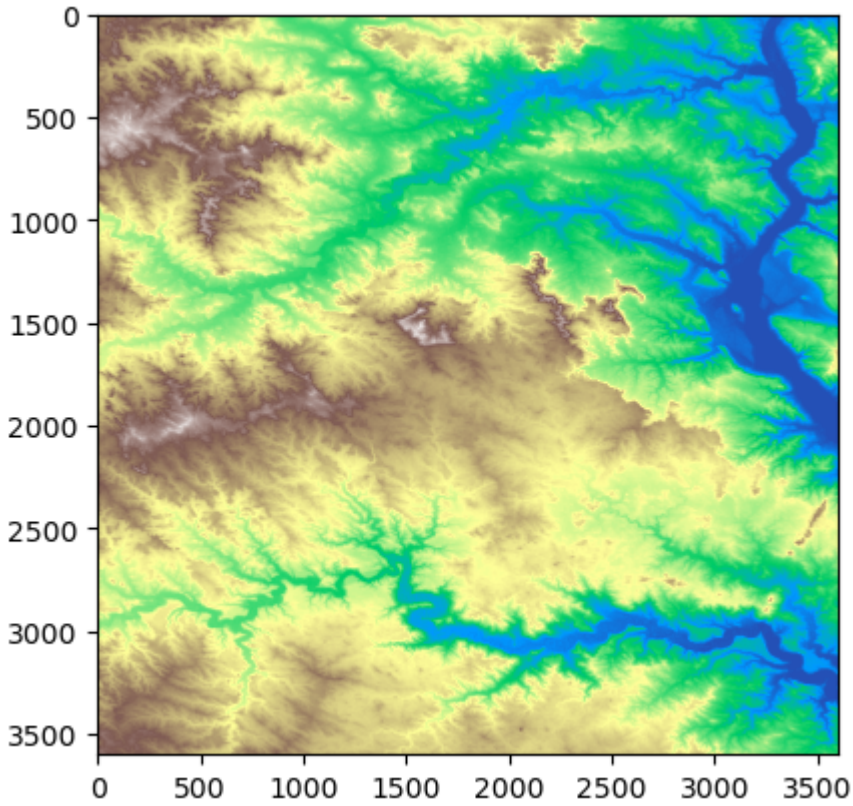
[13]:
```
# create a mosaic from all the images
with rio.Env(aws_session):
    sources = [rio.open(raster) for raster in file_S3]
```

Now we can use the `merge` function to merge these source files together using our list of bounding box values as the bounds. `merge` copies the valid pixels from input rasters and outputs them to a new raster.

[14]:
```
# merge the source files
mosaic, out_trans = merge(sources, bounds = bbox_list)
```

Lastly, we use ma.masked_values for masking all of the NoData values, allowing the mosaic to be plotted correctly. ma.masked_values returns a MaskedArray in which a data array is masked by an approximate value. For the parameters, let's use our mosaic as the data array and the integer of the "nodata" value as the value to mask by. Now we can use `show` to display our masked raster using `matplotlib` with a "terrain" colormap.

[15]:
```
# mask the NoData values so it can be plotted correctly
masked_mosaic = ma.masked_values(mosaic, int(sources[0].nodatavals[0]))

# display the masked mosaic
show(masked_mosaic, cmap = 'terrain')
```



[15]: <Axes: >

3.3.4 Accessing EDAV Data via Web Coverage Service

Authors: Alex Mandel (DevSeed), Samuel Ayers (UAH), Aimee Barciauskas (DevSeed)

Date: April 3, 2023

Description: This example demonstrates how to retrieve raster data from EDAV using a Web Coverage Service (WCS). A WCS lets you access coverage data with multiple dimensions online. The downloaded data is subsetted from data hosted on the MAAP and is available with the full resolution and values.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is recommended to run this tutorial within MAAP's ADE **Pangeo workspace**, which already includes rioxarray.

Additional Resources

- Raster data handling with Python
- OWSLib Github
- rioxarray Documentation

Importing and Installing Packages

We start by installing the libraries that are used to query the WCS connection point, and then to load, explore, and plot the raster data. We use `rasterio` for reading and writing raster formats, `rio-cogeo` for creating and validating Cloud Optimized GEOTIFF (COG) data, and `owslib` for interacting with Open Geospatial Consortium (OGC) services.

[ ]:
```
# install libraries
# %pip is a magic command that installs into the current kernel
# -q means quiet (to give less output)
%pip install -q rasterio
%pip install -q rio-cogeo
%pip install -q owslib
```

After installing the libraries, note that you may see multiple messages that you need to restart the kernel. Then import `rasterio`, `show` from `rasterio.plot` to display images with labeled axes, and `WebCoverageService` from `owslib.wcs` to program with an OGC web service.

[46]:
```
# import rasterio
import rasterio as rio
# import show
from rasterio.plot import show
# import WebCoverageService
from owslib.wcs import WebCoverageService
```

Querying the WCS

Now we can configure the WCS source, use the `getCoverage` function to request a file in GeoTIFF format, and save what is returned to our workspace.

[47]:
```
# configure the WCS source
EDAV_WCS_Base = "https://edav-das.val.esa-maap.org/wcs"
wcs = WebCoverageService(f'{EDAV_WCS_Base}?service=WCS', version='2.0.0')
```

[48]:
```
# request imagery to download
response = wcs.getCoverage(
    identifier=['ESACCI_Biomass_L4_AGB'], # coverage ID
    format='image/tiff', # format what the coverage response will be returned as
    filter='false', # define constraints on query
    scale=1, # resampling factor (1 full resolution, 0.1 resolution degraded of a factor␣
↪of 10)
    subsets=[('Long',11.54,11.8),('Lat',-0.3,0.0)] # subset the image by lat / lon
)

# save the results to file as a tiff
```

(continues on next page)

(continued from previous page)

```
results = "EDAV_example.tif"
with open(results, 'wb') as file:
    file.write(response.read())
```

We can use gdalinfo to provide information about our raster dataset to make sure the data is valid and contains spatial metadata.

[49]:
```
# gives information about the dataset
!gdalinfo {results}
```

```
Driver: GTiff/GeoTIFF
Files: EDAV_example.tif
Size is 293, 339
Warning 1: PROJ: proj_create_from_database: Open of /opt/conda/share/proj failed
Coordinate System is:
GEOGCRS["WGS 84",
    DATUM["World Geodetic System 1984",
        ELLIPSOID["WGS 84",6378137,298.257223563,
            LENGTHUNIT["metre",1,
                ID["EPSG",9001]]]],
    PRIMEM["Greenwich",0,
        ANGLEUNIT["degree",0.0174532925199433,
            ID["EPSG",9122]]],
    CS[ellipsoidal,2],
        AXIS["latitude",north,
            ORDER[1],
            ANGLEUNIT["degree",0.0174532925199433,
                ID["EPSG",9122]]],
        AXIS["longitude",east,
            ORDER[2],
            ANGLEUNIT["degree",0.0174532925199433,
                ID["EPSG",9122]]]]
Data axis to CRS axis mapping: 2,1
Origin = (11.539555555748001,0.000888887639000)
Pixel Size = (0.000888888889000,-0.000888888889000)
Metadata:
  AREA_OR_POINT=Area
  cdm_data_type=INT
  comment=These data were produced at ESA CCI as part of the ESA Biomass CCI project.
  Conventions=CF-1.7
  creator_email=santoro@gamma-rs.ch
  creator_name=GAMMA Remote Sensing
  creator_url=www.gamma-rs.ch
  date_created=2021-6-3
  EPSG=4326
  format_version=CCI Data Standards v2.1
  geospatial_lat_max=0
  geospatial_lat_min=-10
  geospatial_lat_resolution=0.00088889
  geospatial_lat_units=degrees_north
  geospatial_lon_max=20
  geospatial_lon_min=10
  geospatial_lon_resolution=0.00088889
```

(continues on next page)

(continued from previous page)

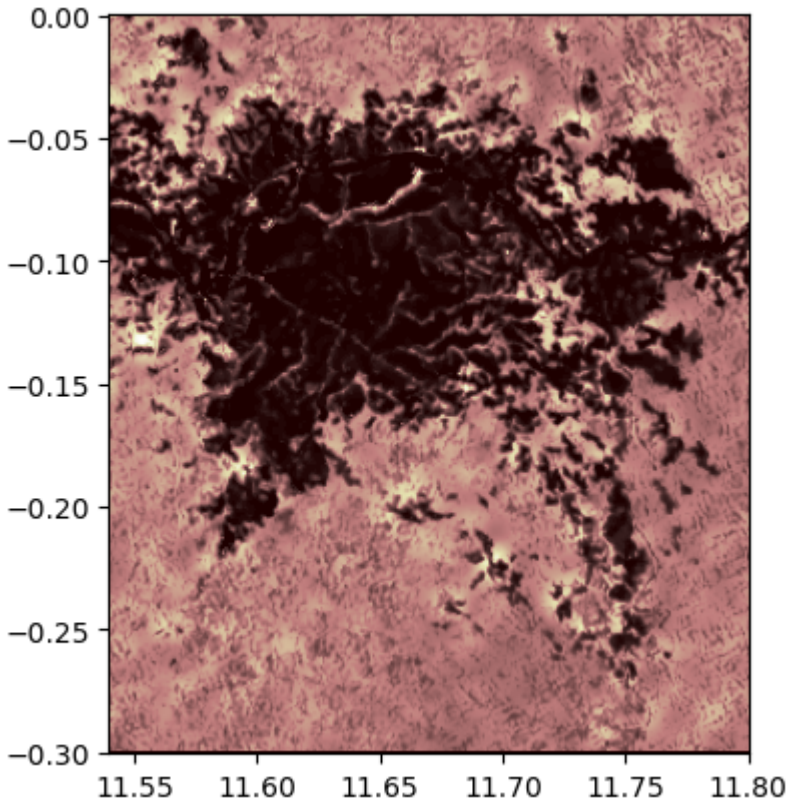
```
  geospatial_lon_units=degrees_east
  geospatial_vertical_max=0.0
  geospatial_vertical_min=0.0
  history=AGB estimation with BIOMASAR-L, v202101, AGB estimation with BIOMASAR-C,␣
↪v202101, Merging of AGB estimates, v202101
  id=N00E010_ESACCI-BIOMASS-L4-AGB_SD-MERGED-100m-${year}-fv3.0.tif
  institution=GAMMA Remote Sensing
  keywords=satellite, observation, forest, biomass
  keywords_vocabulary=NASA Global Change Master Directory (GCMD) Science Keywords
  key_variables=AGB
  license=ESA CCI Data Policy: free and open access
  naming authority=ch.gamma-rs
  platform=ALOS-1 PALSAR, ENVISAT ASAR
  product_version=3.0
  proj4=+proj=longlat +datum=WGS84 +no_defs
  project=Climate Change Initiative - European Space Agency
  references=http://cci.esa.int/biomass
  sensor=PALSAR, SAR-C
  source=ALOS-1 PALSAR FB mosaics, ENVISAT ASAR WideSwath
  spatial_resolution=100 m
  standard_name_vocabulary=NetCDF Climate and Forecast (CF) Metadata Convention version␣
↪67
  summary=This dataset contains a global map of above-ground biomass of the epoch 2010␣
↪obtained from L-and C-band spaceborne SAR backscatter, placed onto a regular grid.
  time_coverage_duration=P1Y
  time_coverage_end=20101231T235959Z
  time_coverage_resolution=P1Y
  time_coverage_start=20100101T000000Z
  title=ESA CCI above-ground biomass standard deviation product level 4, year 2010
  tracking_id=30d2a523-05a8-44ce-8c19-a37b5feb2cf7
Image Structure Metadata:
  COMPRESSION=LZW
  INTERLEAVE=BAND
Corner Coordinates:
Upper Left  (  11.5395556,   0.0008889) ( 11d32'22.40"E,  0d 0' 3.20"N)
Lower Left  (  11.5395556,  -0.3004444) ( 11d32'22.40"E,  0d18' 1.60"S)
Upper Right (  11.8000000,   0.0008889) ( 11d48' 0.00"E,  0d 0' 3.20"N)
Lower Right (  11.8000000,  -0.3004444) ( 11d48' 0.00"E,  0d18' 1.60"S)
Center      (  11.6697778,  -0.1497778) ( 11d40'11.20"E,  0d 8'59.20"S)
Band 1 Block=293x13 Type=UInt16, ColorInterp=Gray
  NoData Value=0
```

Reading the Data

We can now use `rio.open` with our `results` path string and return an opened dataset object. We can set a variable (`rast`) to what is read from this dataset object. Then, we utilize the function `show` to display the raster using Matplotlib.

[50]:
```
# take path and return opened dataset object, set variable to read dataset object
with rio.open(results, 'r') as src:
        rast = src.read()
```

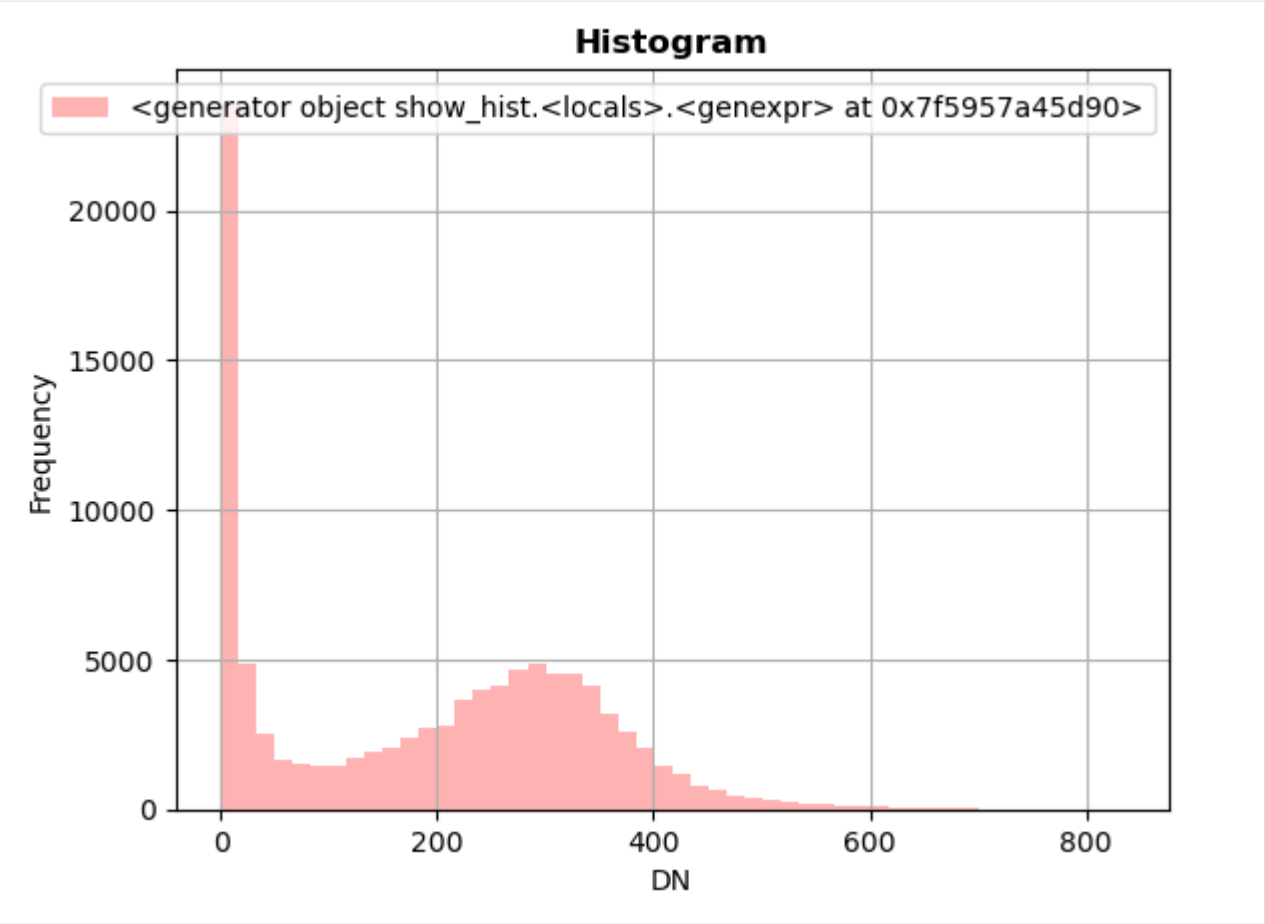
[51]:
```
# make a plot
show(rast, transform=src.transform, cmap='pink')
```



[51]:
```
<AxesSubplot: >
```

We now have a visual of our raster. Let's import and employ the `show_hist` function from `rasterio.plot` to generate a histogram of the raster.

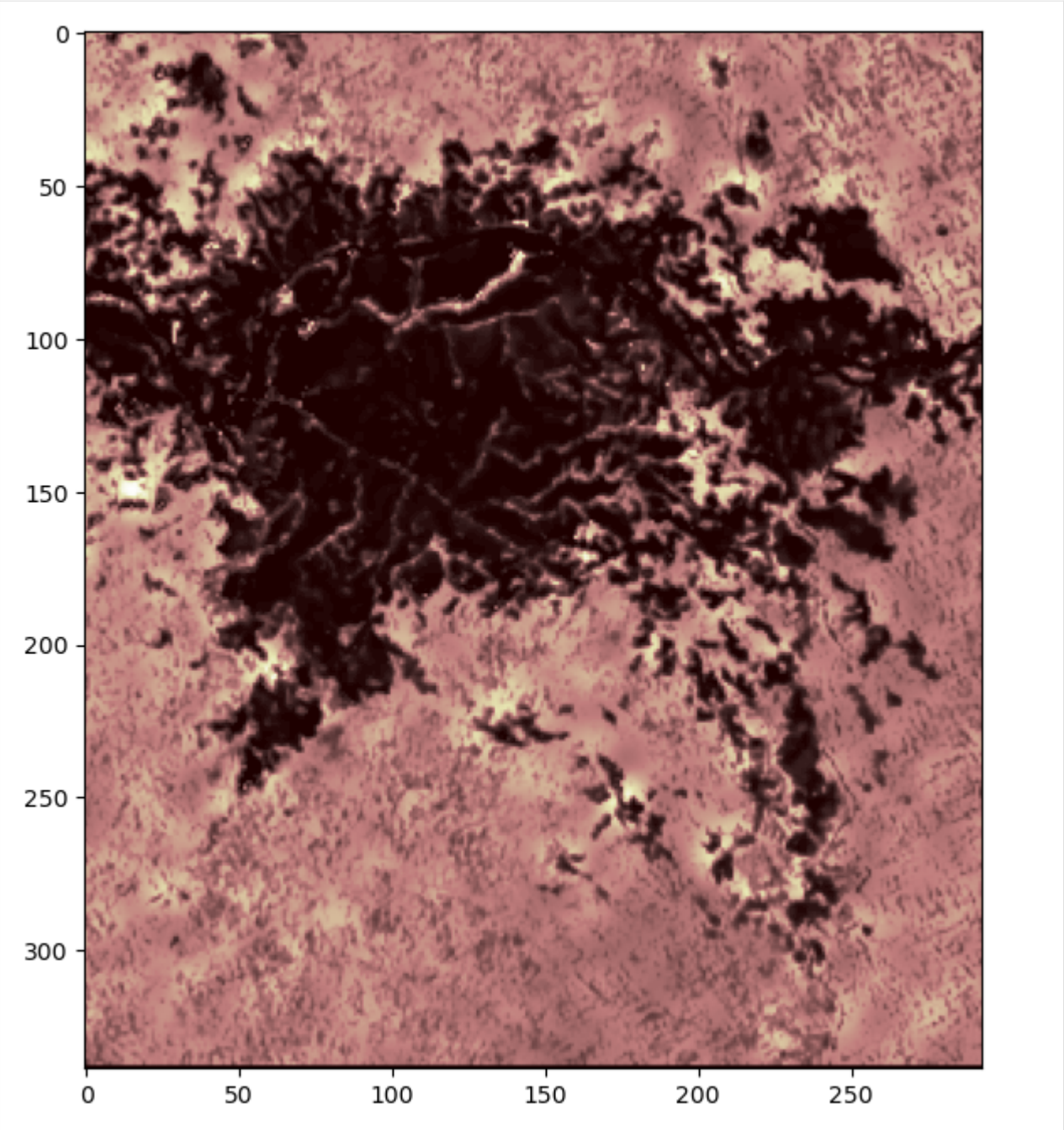
[52]:
```
# import show_hist
from rasterio.plot import show_hist
# create histogram
show_hist(rast,
          bins=50,# number of bins to compute histogram across
          alpha=.3,# transparency
          title="Histogram"# figure title
         )
```



We can also generate a plot using Matplotlib. Let's import `matplotlib.pyplot` and `numpy` and make a new plot. To do this, use the `plt.subplots` function to return a figure and a single "Axes" instance. Then remove single-dimensional entries from the shape of our array using `np.squeeze` and display the data as an image using `imshow`. Now, we can set the norm limits for image scaling using the `set_clim` function.

[53]:
```
# import matplotlib.pyplot
import matplotlib.pyplot as plt
# import numpy
import numpy as np
```

[54]:
```
# set figure and single "Axes" instance
fig, ax = plt.subplots(1, figsize=(8,8))
# remove single-dimensional entries from the shape of the variable rast
#     and display the image
edavplot = ax.imshow(np.squeeze(rast), cmap='pink')
```


0
50
100
150
200
250
300
0
50
100
150
200
250

Newer Method `rioxarray`

Another way to work with raster data is with the `rasterio` "xarray" extension. Let's install and import `rioxarray` and create a plot using the `open_rasterio` and `plot` functions.

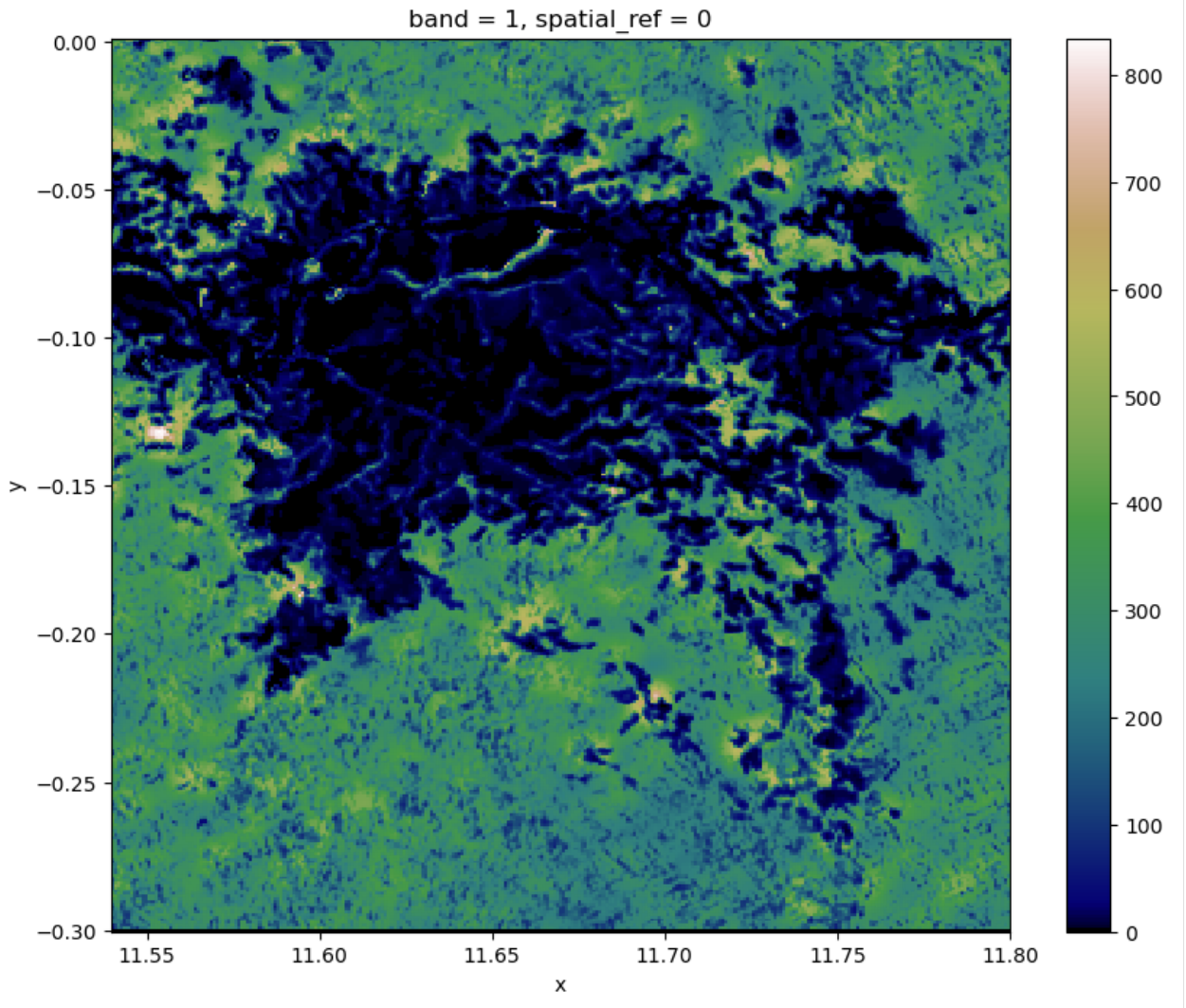
[ ]:
```
# install rasterio xarray extension
%pip install -q rioxarray
```

[56]:
```
# import rasterio xarray extension
import rioxarray
```

[57]:
```
# opens results with rasterio to set dataarray
edav_x = rioxarray.open_rasterio(results)
```

[58]:
```
# plot dataarray
edav_x.plot(cmap="gist_earth", figsize=(10,8))
```

[58]:
```
<matplotlib.collections.QuadMesh at 0x7f591617ae60>
```



3.3.5 Direct Access to LPDAAC GEDI Products

Authors: Alex Mandel (Development Seed), Brian Freitag (NASA MSFC), Jamison French (Development Seed)

Description: In this tutorial, we demonstrate how to use transform HTTPS links into their corresponding S3 links to retrieve GEDI data hosted by the Land Processes Distributed Active Archive Center (LP DAAC).

This tutorial demonstrates a temporary workaround with the expectation that direct access links for LPDAAC GEDI data will eventually be available through NASA CMR.

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Additional Resources

- *Searching Granules in CMR*
- *Searching Collections in CMR*

Importing Packages

If the packages below are not installed already, uncomment the following cell

[ ]:
```
# %pip install h5py fsspec s3fs --quiet
```

[2]:
```
import h5py
import boto3
import botocore
import fsspec
from maap.maap import MAAP

maap = MAAP(maap_host="api.maap-project.org")
```

Searching the Data

We'll start by gathering a sample list of granules from the GEDI L2A collection. The HTTPS links we're after are nested within the granule object.

[3]:
```
results = maap.searchGranule(
    concept_id="C1908348134-LPDAAC_ECS",  # GEDI-L2A
    cmr_host="cmr.earthdata.nasa.gov",
    limit=10,
)

# Download URL of GEDI L2A product
print(results[0].getDownloadUrl())
```

```
https://e4ftl01.cr.usgs.gov//GEDI_L1_L2/GEDI/GEDI02_A.002/2019.04.18/GEDI02_A_
↪2019108002012_O01959_01_T03909_02_003_01_V002.h5
```

Converting the Paths

We'll create a helper function to handle the link conversions to AWS S3 links.

[4]:
```
def lpdaac_gedi_https_to_s3(url):
    dir_comps = url.split("/")
    return f"s3://lp-prod-protected/{dir_comps[6]}/{dir_comps[8].strip('.h5')}/{dir_
↪comps[8]}"


# Sample
lpdaac_gedi_https_to_s3(results[0].getDownloadUrl())
```

[4]:
```
's3://lp-prod-protected/GEDI02_A.002/GEDI02_A_2019108002012_O01959_01_T03909_02_003_01_
↪V002/GEDI02_A_2019108002012_O01959_01_T03909_02_003_01_V002.h5'
```

Accessing the Data

We'll fetch temporary S3 credentials for LPDAAC data and then view the BEAM groups of the first GEDI link in our results.

[5]:
```
credentials = maap.aws.earthdata_s3_credentials(
    'https://data.lpdaac.earthdatacloud.nasa.gov/s3credentials'
)

s3 = fsspec.filesystem(
    "s3",
    key=credentials['accessKeyId'],
    secret=credentials['secretAccessKey'],
    token=credentials['sessionToken']
)
```

[6]:
```
with s3.open(lpdaac_gedi_https_to_s3(results[0]._location), "rb") as f:
    gedi_data = h5py.File(f, "r")
    print(gedi_data.keys())
```

```
<KeysViewHDF5 ['BEAM0000', 'BEAM0001', 'BEAM0010', 'BEAM0011', 'BEAM0101', 'BEAM0110',
↪'BEAM1000', 'BEAM1011', 'METADATA']>
```

[ ]:

3.4 Query

3.4.1 Fields Within the Gedi Cal/Val Data

Author: Samuel Ayers (UAH)

Updated: June 30, 2020

The Global Ecosystem Dynamics Investigation (GEDI) Forest Structure and Biomass Database (FSBD) is a collection of field and LiDAR datasets developed to serve a central repository for calibration and validation of the GEDI mission data. The GEDI Cal/Val Field Data collection contains the field data contributions to the FSBD. The database has contributions across the globe for a variety of vegetation types from projects dating back to 2003.

The table below describes the variables found in the files:

variable	units	description
plot	NA	name of plot
subplot	NA	subplot identifier
survey	NA	name of survey event
private	NA	data privacy (1=private, 0=public)
date	NA	date of survey event
region	NA	region of plot location: Eu=Europe; As=Asia; Au=Australia/New Zealand; Af=Africa; SEAsia=Soutl
vegetation	NA	vegetation type where sampled: Sav = Savannah; TropRF = Tropical rainforest; TempRF = Temperate
map	mm	mean annual rainfall
mat	deg	mean annual temperature
pft.modis	NA	plant functional type from the nearest MCD12Q1 product date
pft.name	NA	plant functional type from field descriptions or other ancillary data
wwf.ecoregion	NA	Terrestrial ecoregion from the WWF Olson et al. (2004) terrestrial ecoregions of the globe
latitude	deg	latitude of location where sampled (-90 to 90 deg South to North)
longitude	deg	longitude of location where sampled (-180 to 180 deg West to East)
p.sample	NA	subplot type: 0=independent fixed area plots; 1=variable area plots; 2=multiple fixed area plots for ag
p.stemmap	NA	stem mapped plot (1=TRUE,0=FALSE)
p.origin	NA	origin of the plot, SW has to be interpreted as the lower left corner relative to the orientation
p.orientation	deg	orientation of plot degrees clockwise from UTM map grid north
p.shape	NA	plot shape (E=ellipse, R=rectangle)
p.majoraxis	m	major axis of plot
p.minoraxis	m	minor axis of plot
p.geom	m	plot geometry WKT string
p.epsg	NA	epsg code of the UTM zone used for the tree coordinates
p.area	m2	plot polygon area
p.mindiam	m	minimum tree diameter measured across all subplots
sp.geom	m	subplot geometry WKT string
sp.ix	NA	subplot track index
sp.iy	NA	subplot pulse index
sp.shape	NA	subplot shape (E=ellipse, R=rectangle)
sp.area	m2	subplot area
sp.mindiam	m	minimum tree diameter measured for the subplot
pai	m2/m2	plant area index of vegetation
lai	m2/m2	leaf area index of vegetation
cover	NA	vertically projected canopy cover (1-Pgap)
dft	NA	dominant plant functional type: EA = evergreen angiosperm; DA = deciduous angiosperm; EG = eve
agb	kg	mass of above-ground standing trees and shrubs
agb.valid	NA	valid above ground mass prediction (1=TRUE,0=FALSE)

Table 1

variable	units	description
agb.lower	kg	lower limit of standard error of mass of above-ground standing trees and shrubs
agb.upper	kg	upper limit of standard error of mass of above-ground standing trees and shrubs
agbd.ha	Mg/ha	mass density of above-ground standing trees and shrubs
agbd.ha.lower	Mg/ha	lower limit of standard error of mass density of above-ground standing trees and shrubs
agbd.ha.upper	Mg/ha	upper limit of standard error of mass density of above-ground standing trees and shrubs
sn	n	plot stem number
snd.ha	n/ha	plot stem number density
sba	m2	plot stand basal area
sba.ha	m2/ha	plot stand basal area per hectare
swsg.ba	g/cm3/m2	basal-area-weighted wood specific gravity
h.t.max	m	maximum total height of plants from ground to highest leaf
sp.agb	kg	subplot mass of above-ground standing trees and shrubs
sp.agb.valid	NA	valid subplot above ground mass prediction (1=TRUE,0=FALSE)
sp.agbd.ha	Mg/ha	subplot mass density of above-ground standing trees and shrubs
sp.agbd.ha.lower	Mg/ha	subplot lower limit of standard error of mass density of above-ground standing trees and shrubs
sp.agbd.ha.upper	Mg/ha	subplot upper limit of standard error of mass density of above-ground standing trees and shrubs
sp.sba.ha	m2/ha	subplot stand basal area per hectare
sp.swsg.ba	g/cm3/m2	subplot basal-area-weighted wood specific gravity
sp.h.t.max	m	subplot maximum total height of plants from ground to highest leaf
l.project	NA	name of the lidar project dataset on UMD servers
l.instr	NA	lidar instrument manufacturer and model
l.epsg	NA	epsg code of the corresponding lidar data
l.date	NA	date of the corresponding lidar acquisition
g.fp	NA	geometry collection WKT string of GEDI lidar footprint center locations
tree.date	NA	date of tree measurement
family	NA	latin name of botanical family
species	NA	latin name of species (genus species)
pft	NA	plant functional type: EA = evergreen angiosperm; DA = deciduous angiosperm; EG = evergreen gy
wsg	g/cm3	the ratio of wood density to water
wsg.sd	g/cm3	standard deviation of WSG sample values
tree	NA	tree identifier
stem	NA	stem identifier
x	m	easting UTM coordinate
y	m	northing UTM coordinate
z	m	elevation relative to geoid coordinate
status	NA	live tree (1=TRUE,0=FALSE)
allom.key	NA	key to allometric LUT
a.stem	m2	area of total stem cross-section at measurement height
h.t	m	total height of plant from ground to highest leaf
h.t.mod	NA	tree heights modelled using local or regional DBH-Ht relationship
d.stem	m	diameter of stem at measurement height
d.stem.valid	NA	valid tree diameter to use in AGB calculation (1=TRUE,0=FALSE)
d.ht	m	height at which stem diameter was measured
c.w	m	diameter or width of crown
m.agb	kg	mass of above-ground components of tree

3.5 User Data

3.5.1 Adding Cloud-Optimized GeoTIFFs to the MAAP Biomass Earthdata Dashboard

Author(s): Aimee Barciauskas (Development Seed)

Date: Oct 14, 2021

Description: The following notebook steps through how to add a dataset to the MAAP Dashboard.

Note, there are 2 scenarios:

1. Adding a single Cloud-Optimized GeoTIFF (COG), and
2. Adding many distinct COGs as a "mosaic" with mosaicJSON.

High-level, the steps are:

1. Inspect your Cloud-Optimized GeoTIFF(s) (COGs) to understand the best rescale and colormap name parameters. Optionally create a mosaic.
2. Define a colormap. Colormaps provide mappings of data values to RGB values.
3. Create a PR to the datasets repo to add or update your dataset.

The MAAP dashboard has 3 environments:

1. Developer-in-test (DIT): https://biomass.dit.maap-project.org
2. Staging: https://biomass.staging.maap-project.org
3. Production: https://biomass.maap-project.org

These instructions will guide you towards adding your dataset to `biomass.dit.maap-project.org`. The MAAP Dashboard team will "promote" changes to staging and production periodically (release schedule forthcoming).

Run This Notebook

To access and run this tutorial within MAAP's Algorithm Development Environment (ADE), please refer to the "Getting started with the MAAP" section of our documentation.

Disclaimer: it is highly recommended to run a tutorial within MAAP's ADE, which already includes packages specific to MAAP, such as maap-py. Running the tutorial outside of the MAAP ADE may lead to errors.

Additional Resources

- Rio Tiler Colors
- Matplotlib Colors

Importing and Installing Packages

To be able to run this notebook you'll need the following requirements: - rasterio - rio-cogeo - requests - cogeo-mosaic - folium

If the packages below are not installed already, uncomment the following cell:

[1]:
```
# %pip install rasterio
# %pip install rio-cogeo
# %pip install requests
# %pip install cogeo-mosaic
# %pip install folium
```

[2]:
```
import glob
import json
import os
import matplotlib

import requests
from pprint import pprint
from cogeo_mosaic.mosaic import MosaicJSON
from cogeo_mosaic.backends import MosaicBackend
from folium import Map, TileLayer, WmsTileLayer

titiler_endpoint = "https://titiler.maap-project.org"
```

Step 1: Inspect Cloud-Optimized GeoTIFF(s)

In this step, we ensure that our data is valid, accessible, and looks as expected. We've included a helper function to translate MAAP ADE local paths to their respective S3 urls.

Accessing Files

[3]:
```
project_dir = "/projects/shared-buckets/<your_name>/<project_dir>"

# e.g.
project_dir = "/projects/shared-buckets/alexdevseed/landsat8/viz/"
```

[4]:
```
# Search for files to include, use recursive if nested folders (common in DPS output)
files = glob.glob(os.path.join(project_dir, "Landsat8*.tif"), recursive=False)
files = [os.path.basename(f) for f in files]
pprint(files[:10])

# Use the first product
_tif = files[0]


# Helper function
def local_to_s3(url):
    """A Function to convert local paths to s3 urls"""
    return url.replace("/projects/shared-buckets", "s3://maap-ops-workspace/shared")
```

```
['Landsat8_30542_comp_cog_2015-2020_dps.tif',
 'Landsat8_30543_comp_cog_2015-2020_dps.tif',
 'Landsat8_30822_comp_cog_2015-2020_dps.tif',
 'Landsat8_30823_comp_cog_2015-2020_dps.tif']
```

[5]:
```
%%bash -s "$project_dir" "$_tif"
rio cogeo validate $1/$2
```

```
/projects/shared-buckets/alexdevseed/landsat8/viz/Landsat8_30542_comp_cog_2015-2020_dps.
↪tif is a valid cloud optimized GeoTIFF
```

[6]:
```
# Getting COG information
cog_info = requests.get(
    f"{titiler_endpoint}/cog/info",
    params={
        "url": f"{local_to_s3(project_dir)}{_tif}",
    },
).json()

bounds = cog_info["bounds"]
pprint(cog_info)
```

```
{'band_descriptions': [['b1', 'Blue'],
                       ['b2', 'Green'],
                       ['b3', 'Red'],
                       ['b4', 'NIR'],
                       ['b5', 'SWIR'],
                       ['b6', 'NDVI'],
                       ['b7', 'SAVI'],
                       ['b8', 'MSAVI'],
                       ['b9', 'NDMI'],
                       ['b10', 'EVI'],
                       ['b11', 'NBR'],
                       ['b12', 'NBR2'],
                       ['b13', 'TCB'],
                       ['b14', 'TCG'],
                       ['b15', 'TCW'],
                       ['b16', 'ValidMask'],
                       ['b17', 'Xgeo'],
                       ['b18', 'Ygeo']],
 'band_metadata': [['b1', {}],
                   ['b2', {}],
                   ['b3', {}],
                   ['b4', {}],
                   ['b5', {}],
                   ['b6', {}],
                   ['b7', {}],
                   ['b8', {}],
                   ['b9', {}],
                   ['b10', {}],
                   ['b11', {}],
                   ['b12', {}],
                   ['b13', {}],
```

(continues on next page)

(continued from previous page)

```
                  ['b14', {}],
                  ['b15', {}],
                  ['b16', {}],
                  ['b17', {}],
                  ['b18', {}]],
 'bounds': [-117.10749852280769,
            50.78795362739066,
            -116.50936927974429,
            51.16389512140189],
 'colorinterp': ['gray',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined',
                 'undefined'],
 'count': 18,
 'driver': 'GTiff',
 'dtype': 'float32',
 'height': 1000,
 'maxzoom': 12,
 'minzoom': 9,
 'nodata_type': 'None',
 'overviews': [2, 4],
 'width': 1000}
```

[7]:
```
# Getting band information
cog_stats = requests.get(
    f"{titiler_endpoint}/cog/statistics",
    params={
        "url": f"{local_to_s3(project_dir)}{_tif}",
    },
).json()
pprint(cog_stats["b1"])
```

```
{'count': 1000000.0,
 'histogram': [[128480.0,
                807237.0,
                53374.0,
                7274.0,
                2056.0,
```

(continues on next page)

(continued from previous page)

```
                743.0,
                370.0,
                299.0,
                129.0,
                38.0],
               [0.0,
                4814.5,
                9629.0,
                14443.5,
                19258.0,
                24072.5,
                28887.0,
                33701.5,
                38516.0,
                43330.5,
                48145.0]],
 'majority': 0.0,
 'masked_pixels': 0.0,
 'max': 48145.0,
 'mean': 7467.137024,
 'median': 7989.0,
 'min': 0.0,
 'minority': 11663.0,
 'percentile_2': 0.0,
 'percentile_98': 12191.0,
 'std': 2803.86812414882,
 'sum': 7467137024.0,
 'unique': 20318.0,
 'valid_percent': 100.0,
 'valid_pixels': 1000000.0}
```

Create Parameters for the TiTiler

These parameters will be pased to `titiler_endpoint` for visualization.

Note the values below: We're setting the `rescale` equal to the selected band's `min`,`max` values and selecting the `gist_earth_r` colormap. You should modify the `colormap_name` as makes sense for your dataset. This notebook includes a section on what colormaps are available and how to configure different types of colormaps and legends.

[8]:
```
band = "b1"
bidx = 1
rescale = f"{cog_stats[band]['min']},{cog_stats[band]['max']}"

params = {
    "tile_format": "png",
    "tile_scale": "1",
    "TileMatrixSetId": "WebMercatorQuad",
    "url": f"{local_to_s3(project_dir)}{_tif}",
    "bidx": 1,  # Select which band to use
    "resampling": "nearest",
    "rescale": rescale,
```

(continues on next page)

(continued from previous page)

```
    "return_mask": "true",
    "colormap_name": "gist_earth_r",
}
```

Scenario 1: Adding a Single COG

Upload File

Only use the following steps if you only have one COG to share to the dashboard. If you want to create a mosaic from multiple COGs, skip Scenario 1 and go to Scenario 2.

If you haven't already, upload the file to S3 and make note of the location. In this tutorial, we're using a landsat8/visualization TIF already in S3 for the `url` parameter value.

Test COG with TiTiler and Folium

[9]:
```
response = requests.get(f"{titiler_endpoint}/cog/tilejson.json", params=params).json()
```

[10]:
```
m = Map(
    tiles="OpenStreetMap",
    location=((bounds[1] + bounds[3]) / 2, (bounds[0] + bounds[2]) / 2),
    zoom_start=cog_info["minzoom"],
)

tiles = TileLayer(tiles=response["tiles"][0], opacity=1, attr="USGS")

tiles.add_to(m)
m
```

[10]:
```
<folium.folium.Map at 0x7f78ef4be440>
```

Scenario 2: Adding Data from Multiple COGs by Creating a Mosaic

Many datasets are comprised of many tiles distributed spatially over the globe. In order to visualize them all together, we can use mosaicJSON to create a mosaic for the dynamic tiler API. The dynamic tiler API knows how to read this mosaicJSON and select which tiles to render based on the current zoom, x and y coordinates across spatially distinct COGs.

[11]:
```
tiles = [f"{local_to_s3(project_dir)}{file}" for file in files]
```

[12]:
```
mosaicdata = MosaicJSON.from_urls(tiles, minzoom=1, maxzoom=16)
mosaicdata
```

[12]:
```
MosaicJSON(mosaicjson='0.0.3', name=None, description=None, version='1.0.0',␣
↪attribution=None, minzoom=1, maxzoom=16, quadkey_zoom=1, bounds=(-117.19773367251135,␣
↪50.19386902261471, -116.26013039328576, 51.16389512140189), center=(-116.
↪72893203289856, 50.67888207200831, 1), tiles={'0': ['s3://maap-ops-workspace/shared/
↪alexdevseed/landsat8/viz/Landsat8_30542_comp_cog_2015-2020_dps.tif', 's3://maap-ops-
```

(continues on next page)

(continued from previous page)

```
↪workspace/shared/alexdevseed/landsat8/viz/Landsat8_30543_comp_cog_2015-2020_dps.tif',
↪'s3://maap-ops-workspace/shared/alexdevseed/landsat8/viz/Landsat8_30822_comp_cog_2015-
↪2020_dps.tif', 's3://maap-ops-workspace/shared/alexdevseed/landsat8/viz/Landsat8_30823_
↪comp_cog_2015-2020_dps.tif']}, tilematrixset=None, asset_type=None, asset_prefix=None,␣
↪data_type=None, colormap=None, layers=None)
```

Using MosaicJSON with TiTiler

There are 2 options for using mosaicJSON with titiler:

1. (Preferred) Post mosaicJSON to titiler `mosaics` endpoint and use the `mosaicjson/mosaics` endpoint for dynamic tiling.
2. Upload mosaicJSON to S3 and pass the S3 url to the titiler `mosaicjson/tiles` endpoint.

Post MosaicJSON to TiTiler

[13]:
```
mosaic_links = requests.post(
    url=f"{titiler_endpoint}/mosaics",
    headers={
        "Content-Type": "application/vnd.titiler.mosaicjson+json",
    },
    json=mosaicdata.model_dump(exclude_none=True),
).json()

pprint(mosaic_links)

mosaic_id = mosaic_links["id"]
```

```
{'id': '1cecf064-1f2c-4adf-b7b3-7121fdc8f97d',
 'links': [{'href': 'https://titiler.maap-project.org/mosaics/1cecf064-1f2c-4adf-b7b3-
↪7121fdc8f97d',
            'rel': 'self',
            'title': 'Self',
            'type': 'application/json'},
           {'href': 'https://titiler.maap-project.org/mosaics/1cecf064-1f2c-4adf-b7b3-
↪7121fdc8f97d/mosaicjson',
            'rel': 'mosaicjson',
            'title': 'MosaicJSON',
            'type': 'application/json'},
           {'href': 'https://titiler.maap-project.org/mosaics/1cecf064-1f2c-4adf-b7b3-
↪7121fdc8f97d/tilejson.json',
            'rel': 'tilejson',
            'title': 'TileJSON',
            'type': 'application/json'},
           {'href': 'https://titiler.maap-project.org/mosaics/1cecf064-1f2c-4adf-b7b3-
↪7121fdc8f97d/tiles/{z}/{x}/{y}',
            'rel': 'tiles',
            'title': 'Tiles',
            'type': 'application/json'},
           {'href': 'https://titiler.maap-project.org/mosaics/1cecf064-1f2c-4adf-b7b3-
```

(continues on next page)

(continued from previous page)

```
↪7121fdc8f97d/WMTSCapabilities.xml',
            'rel': 'wmts',
            'title': 'WMTS',
            'type': 'application/json'}]}
```

[14]:
```
tilejson_endpoint = list(
    filter(lambda x: x.get("rel") == "tilejson", dict(mosaic_links)["links"])
)
tilejson_endpoint
```

[14]:
```
[{'href': 'https://titiler.maap-project.org/mosaics/1cecf064-1f2c-4adf-b7b3-7121fdc8f97d/
↪tilejson.json',
  'rel': 'tilejson',
  'type': 'application/json',
  'title': 'TileJSON'}]
```

Test Mosaic with TiTiler and Folium

[15]:
```
params = {
    "tile_format": "png",
    "bidx": bidx,
    "resampling": "nearest",
    "rescale": rescale,
    "return_mask": "true",
    "colormap_name": "viridis",
    "pixel_selection": "first",
}

r_te = requests.get(tilejson_endpoint[0]["href"], params=params).json()

tiles = TileLayer(tiles=f"{r_te['tiles'][0]}", opacity=1, attr="USGS")

tiles.add_to(m)
m
```

[15]:
```
<folium.folium.Map at 0x7f78ef4be440>
```

Step 2: Define a Color Map

By default, the image will be displayed in greyscale if no `colormap_name` parameter is passed to the titiler API. Guidance below is provided to help determine what a valid colormap_name might be and how to create a legend for the dashboard.

Dashboard ColorRamps & Legends

When using the dashboard, there 2 components for implementing a color scheme for your map. There is the map render and there is the legend.

> Titiler used for Cloud Optimized Geotiff (COG) rendering accepts any color scheme from the python matplotlib library, and custom color formulas.

- Rio Tiler Colors
- Matplotlib Colors

Available `colormap_name` values for titiler: `above, accent, accent_r, afmhot, afmhot_r, autumn, autumn_r, binary, binary_r, blues, blues_r, bone, bone_r, brbg, brbg_r, brg, brg_r, bugn, bugn_r, bupu, bupu_r, bwr, bwr_r, cfastie, cividis, cividis_r, cmrmap, cmrmap_r, cool, cool_r, coolwarm, coolwarm_r, copper, copper_r, cubehelix, cubehelix_r, dark2, dark2_r, flag, flag_r, gist_earth, gist_earth_r, gist_gray, gist_gray_r, gist_heat, gist_heat_r, gist_ncar, gist_ncar_r, gist_rainbow, gist_rainbow_r, gist_stern, gist_stern_r, gist_yarg, gist_yarg_r, gnbu, gnbu_r, gnuplot, gnuplot2, gnuplot2_r, gnuplot_r, gray, gray_r, greens, greens_r, greys, greys_r, hot, hot_r, hsv, hsv_r, inferno, inferno_r, jet, jet_r, magma, magma_r, nipy_spectral, nipy_spectral_r, ocean, ocean_r, oranges, oranges_r, orrd, orrd_r, paired, paired_r, pastel1, pastel1_r, pastel2, pastel2_r, pink, pink_r, piyg, piyg_r, plasma, plasma_r, prgn, prgn_r, prism, prism_r, pubu, pubu_r, pubugn, pubugn_r, puor, puor_r, purd, purd_r, purples, purples_r, rainbow, rainbow_r, rdbu, rdbu_r, rdgy, rdgy_r, rdpu, rdpu_r, rdylbu, rdylbu_r, rdylgn, rdylgn_r, reds, reds_r, rplumbo, schwarzwald, seismic, seismic_r, set1, set1_r, set2, set2_r, set3, set3_r, spectral, spectral_r, spring, spring_r, summer, summer_r, tab10, tab10_r, tab20, tab20_r, tab20b, tab20b_r, tab20c, tab20c_r, terrain, terrain_r, twilight, twilight_r, twilight_shifted, twilight_shifted_r, viridis, viridis_r, winter, winter_r, wistia, wistia_r, ylgn, ylgn_r, ylgnbu, ylgnbu_r, ylorbr, ylorbr_r, ylorrd, ylorrd_r`

Example 1: Class Based Known Colors

In this example, the raster represents classes of forest with 11 possible values. There are specific colors selected to correspond to each class. We combine the list of colors and the list of classes and format them for the legend parameter the dashboard needs.

https://github.com/MAAP-Project/dashboard-datasets-maap/blob/main/datasets/taiga-forest-classification.json

[16]:
```
colors = [
    "#5255A3",
    "#1796A3",
    "#FDBF6F",
    "#FF7F00",
    "#FFFFBF",
    "#D9EF8B",
    "#91CF60",
    "#1A9850",
```

(continues on next page)

(continued from previous page)

```
    "#C4C4C4",
    "#FF0000",
    "#0000FF",
]
labels = [
    "Sparse & Uniform",
    "Sparse & Diffuse-gradual",
    "Sparse & Diffuse-rapid",
    "Sparse & Abrupt ",
    "Open & Uniform ",
    "Open & Diffuse-gradual",
    "Open & Diffuse-rapid",
    "Open & Abrupt",
    "Intermediate & Closed",
    "Non-forest edge (dry)",
    "Non-forest edge (wet)",
]

legend = [dict(color=colors[i], label=labels[i]) for i in range(0, len(colors))]
print(json.dumps(legend, indent=2))

# Copy and Paste the output below to your dashboard config.
```

```
[
  {
    "color": "#5255A3",
    "label": "Sparse & Uniform"
  },
  {
    "color": "#1796A3",
    "label": "Sparse & Diffuse-gradual"
  },
  {
    "color": "#FDBF6F",
    "label": "Sparse & Diffuse-rapid"
  },
  {
    "color": "#FF7F00",
    "label": "Sparse & Abrupt "
  },
  {
    "color": "#FFFFBF",
    "label": "Open & Uniform "
  },
  {
    "color": "#D9EF8B",
    "label": "Open & Diffuse-gradual"
  },
  {
    "color": "#91CF60",
    "label": "Open & Diffuse-rapid"
  },
```

(continues on next page)

(continued from previous page)

```
  {
    "color": "#1A9850",
    "label": "Open & Abrupt"
  },
  {
    "color": "#C4C4C4",
    "label": "Intermediate & Closed"
  },
  {
    "color": "#FF0000",
    "label": "Non-forest edge (dry)"
  },
  {
    "color": "#0000FF",
    "label": "Non-forest edge (wet)"
  }
]
```

Example 2: Discrete ColorRamp

In this example, the range of values is known, but the color scale has many non-sequential colors. Starting with the premade color list, we create a continuous color ramp that uses the known colors as stops points. Arbitrarly 12 breaks looked decent in the dashboard legend so we split it into 12 discrete colors. Then combine the list of values and colors into the correct json syntax.

https://github.com/MAAP-Project/dashboard-datasets-maap/blob/main/datasets/ATL08.json

[17]:
```
forest_ht = matplotlib.colors.LinearSegmentedColormap.from_list(
    "forest_ht",
    [
        "#636363",
        "#FC8D59",
        "#FEE08B",
        "#FFFFBF",
        "#D9EF8B",
        "#91CF60",
        "#1A9850",
        "#005A32",
    ],
    12,
)
cols = [matplotlib.colors.to_hex(forest_ht(i)) for i in range(forest_ht.N)]

cats = range(0, 25, (25 // len(cols)))
legend = [[cats[i], cols[i]] for i in range(0, len(cols))]
text = json.dumps(legend, separators=(",", ": "))

print(text.replace("],[", "],\n["))

# Copy and Paste the output below to your dashboard config.
```

```
[[0,"#636363"],
[2,"#c47e5d"],
[4,"#fda467"],
[6,"#fed886"],
[8,"#fff1a7"],
[10,"#f8fcb6"],
[12,"#e0f294"],
[14,"#b8e077"],
[16,"#86ca5f"],
[18,"#3aa754"],
[20,"#118145"],
[22,"#005a32"]]
```

Example 3: Continuous ColorRamp

In this example, we are using a built in ColorRamp from matplotlib. So we just need to extract enough colors to fill the legend adequately, and convert the colors to hex codes.

https://github.com/MAAP-Project/dashboard-datasets-maap/blob/main/datasets/topo.json

[18]:
```
cmap_name = "gist_earth_r"
cmap = matplotlib.cm.get_cmap(cmap_name, 12)
cols = [matplotlib.colors.to_hex(cmap(i)) for i in range(cmap.N)]
print(cols)

# Copy and Paste the output below to your dashboard config.
```

```
['#fdfbfb', '#e3c3b5', '#c9a87a', '#bab060', '#9db059', '#76a652', '#45994a', '#3a8c66',
↪'#2e7c7f', '#1f567b', '#0f2577', '#000000']
```

Step 3: Create and Submit Pull Request to Add Dashboard Dataset

[19]:
```
# This example is for a continuous color ramps
dataset_type = "raster"
dataset_id = "paraguay-estimated-biomass"
dataset_name = "Estimated Biomass in Paraguay"

stops = cols
legend_type = "gradient-adjustable"
info = "Estimated biomass within 6km grids."

sample_bidx = 1
sample_band_min = 0
sample_band_max = 4000
parameters = (
    f"colormap_name={cmap_name}&rescale={sample_band_min},{sample_band_max}&bidx={bidx}"
)
```

[20]:
```
# Single COG
tiles_link = f"{titiler_endpoint}/cog/tiles/{{z}}/{{x}}/{{y}}.png?url=s3://example-
↪bucket/path/to/object/example.tif&{parameters}"
```

(continues on next page)

(continued from previous page)

```
# Mosaic
mosaic_link = (
    f"{titiler_endpoint}/mosaic/{mosaic_id}/tiles/{{z}}/{{x}}/{{y}}?{parameters}"
)
```

[21]:
```
dataset_dict = {
    "id": dataset_id,
    "name": dataset_name,
    "type": dataset_type,
    "swatch": {"color": "#6976d7", "name": "Moody Blue"},
    "source": {"type": dataset_type, "tiles": [tiles_link]},
    "legend": {
        "type": legend_type,
        "min": sample_band_min,
        "max": sample_band_max,
        "stops": stops,
    },
    "info": info,
}
print(json.dumps(dataset_dict, indent=4))
```

```
{
    "id": "paraguay-estimated-biomass",
    "name": "Estimated Biomass in Paraguay",
    "type": "raster",
    "swatch": {
        "color": "#6976d7",
        "name": "Moody Blue"
    },
    "source": {
        "type": "raster",
        "tiles": [
            "https://titiler.maap-project.org/cog/tiles/{z}/{x}/{y}.png?url=s3://example-
↪bucket/path/to/object/example.tif&colormap_name=gist_earth_r&rescale=0,4000&bidx=1"
        ]
    },
    "legend": {
        "type": "gradient-adjustable",
        "min": 0,
        "max": 4000,
        "stops": [
            "#fdfbfb",
            "#e3c3b5",
            "#c9a87a",
            "#bab060",
            "#9db059",
            "#76a652",
            "#45994a",
            "#3a8c66",
            "#2e7c7f",
            "#1f567b",
```

(continues on next page)

(continued from previous page)

```
            "#0f2577",
            "#000000"
        ]
    },
    "info": "Estimated biomass within 6km grids."
}
```

Clone the Datasets Repository

```
git clone git@github.com:MAAP-Project/biomass-dashboard-datasets.git
cd biomass-dashboard-datasets
git checkout -b feature/dataset-name
# select and copy json above
echo <copied_json> >> datasets/paraguay-estimated-biomass.json
```

Add JSON to Product or Country Pilot

In `country_pilots/paraguay/country_pilot.json`:

```
{
    "id": "paraguay",
    "label": "Paraguay",
    //...
    "datasets": [
        {
            "id": "paraguay-forest-mask"
        },
        {
            "id": "paraguay-tree-cover"
        },
        {
            "id": "paraguay-estimated-biomass"
        }
    ]
}
```

Add Content to summary.html

There should be a `summary.html` file corresponding to the product or country pilot you are working on, for example: `country_pilots/paraguay/summary.html`. Add or modify content in that file as appropriate.

Add Dataset(s) to config.yml

In `config.yml`:

```
DATASETS:
- paraguay-estimated-biomass.json
```

Create Pull Request

Once you have added the dataset json file and summary content, submit a PR to https://github.com/MAAP-Project/biomass-dashboard-datasets. A member of the data team will review the PR and when it is merged your content will appear in biomass.dit.maap-project.org.

CHAPTER

FOUR

SYSTEM REFERENCE GUIDE

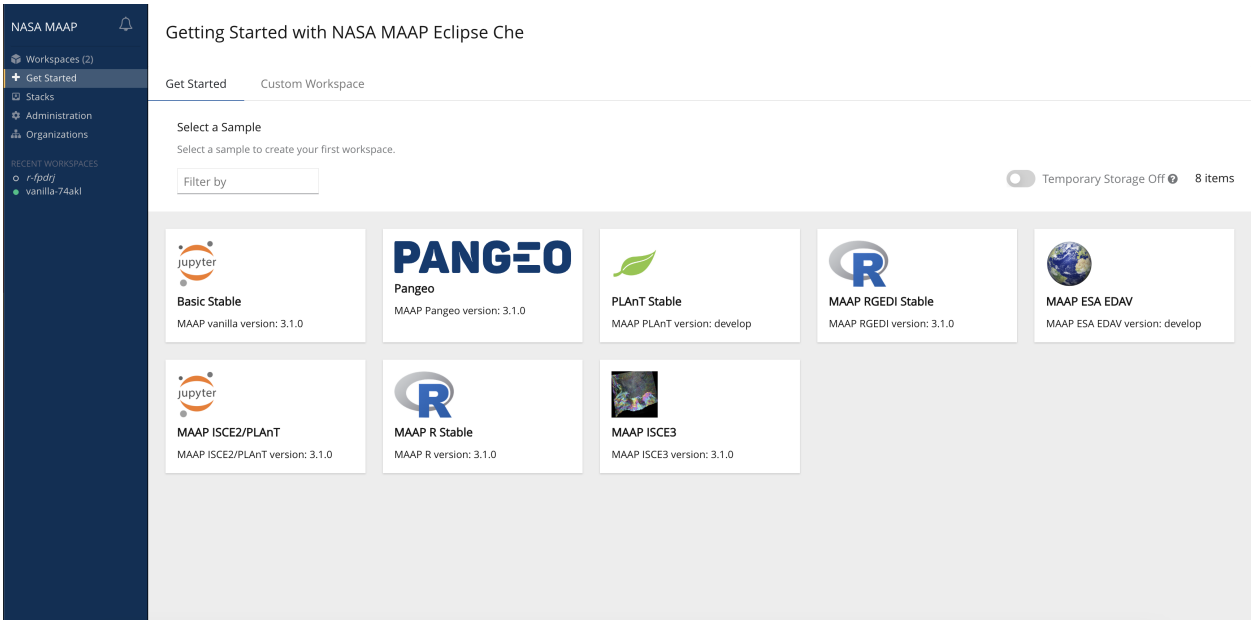
4.1 Workspace Creation and Set-up

4.1.1 Create a Basic Workspace

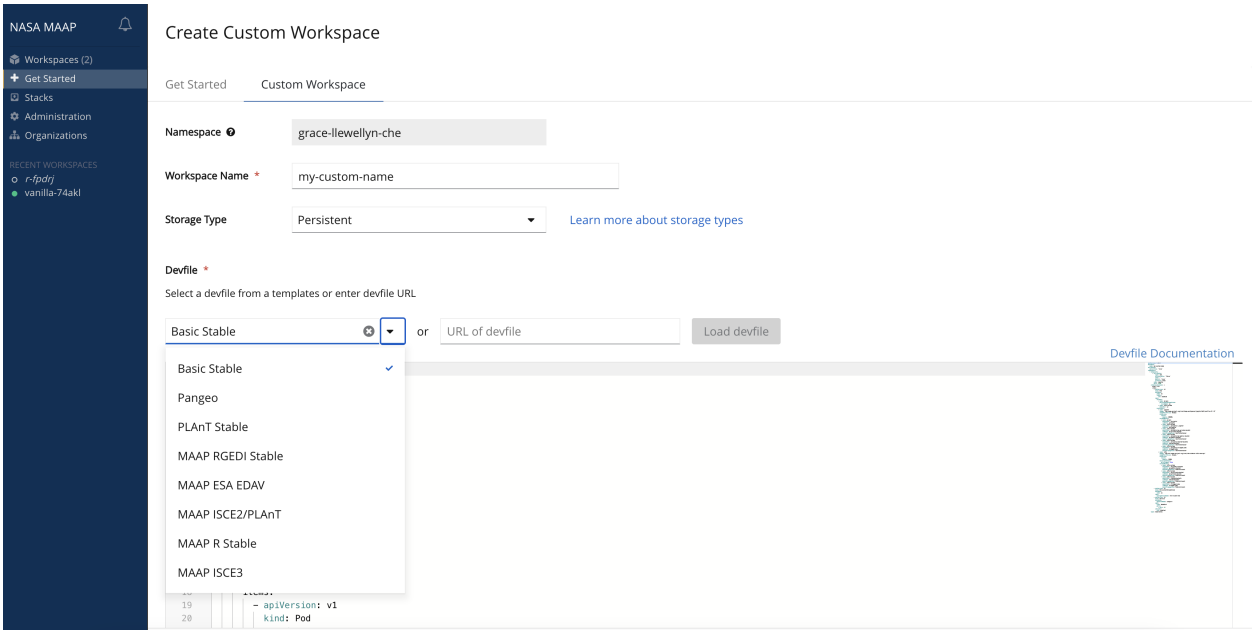
Navigate to the side panel tab `Workspaces` and click `Add Workspace`. On the Get Started tab, you can select a Stack (pre-set environment) to get started quickly.

To see what each Stack has pre-installed, check the Github repo (https://github.com/MAAP-Project/maap-workspaces/tree/main/base_images) and look at the Dockerfiles. Currently there is not a better way to reliably understand what is in each.

To customize your Workspace after creating it, you can use conda. Further documentation on how to do this is in the *Environments section* of this documentation.

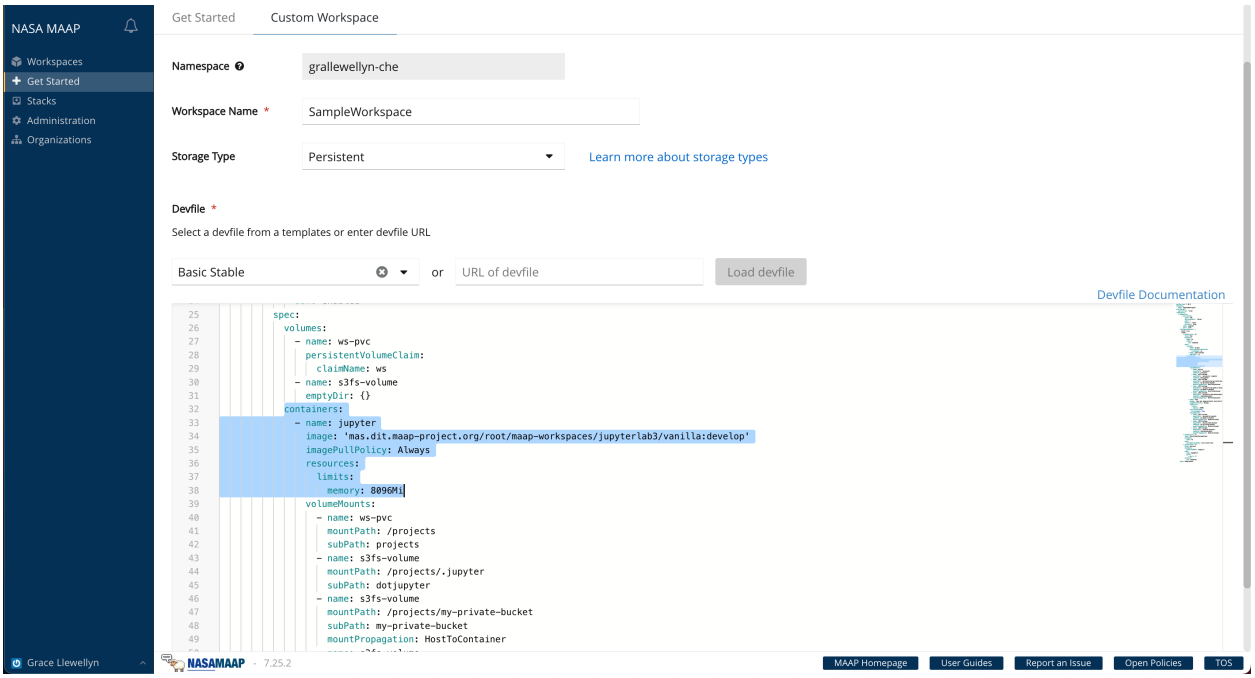


If you would like a more customized approach, on the Custom Workspace tab you can name your workspace, select the storage type, and select a devfile template from available stacks or upload your own devfile.



4.1.2 Advanced Creation: Alter the Workspace Memory Limit

Here, you can edit the devfile under `containers/jupyter/resources/limits/memory` to alter the memory limit for your workspace.



If you want to edit the devfile or your workspace's memory limit after its creation, just click on the `Workspaces` side panel tab, then the workspace, then the Devfile tab at the top of the page.

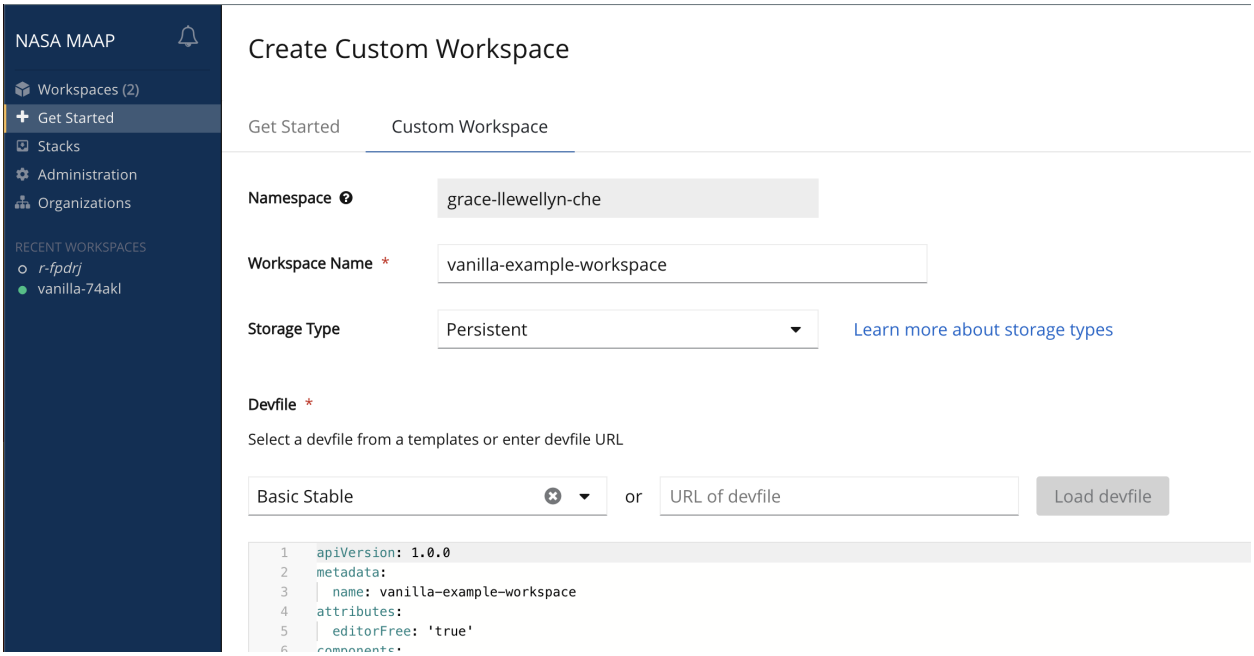
DISCLAIMER: MAAP might change the way that we do workspace memory limits in the future, including restricting the memory that a user can allocate for their workspace. For now, as a courtesy to other users only increase the memory limit if your kernel keeps running out of memory and crashing. If you're not sure why your kernel is crashing after

increasing your workspace's memory limit, please contact the development team. We recommend using a memory limit of 16GB (which means altering the devfile to `memory: 15258Mi`).

4.1.3 How Do I Rename My Workspace?

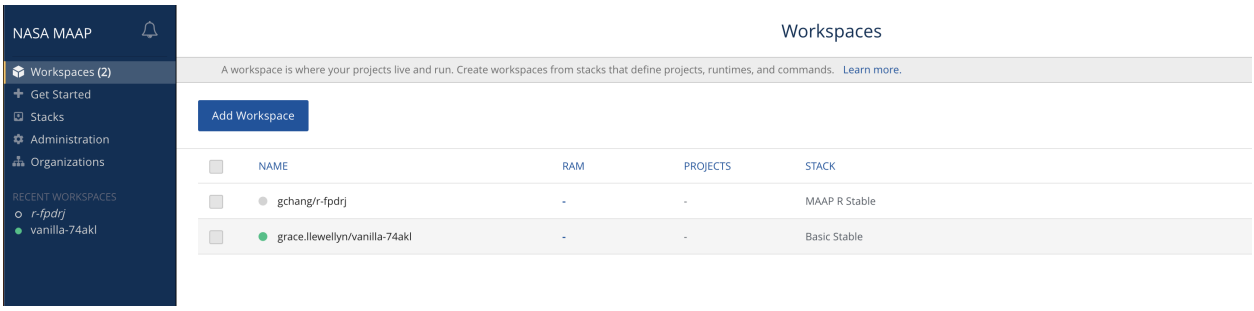
Option 1: At Workspace Creation

You can replace the auto-generated workspace name during creation via the Custom Workspace option.

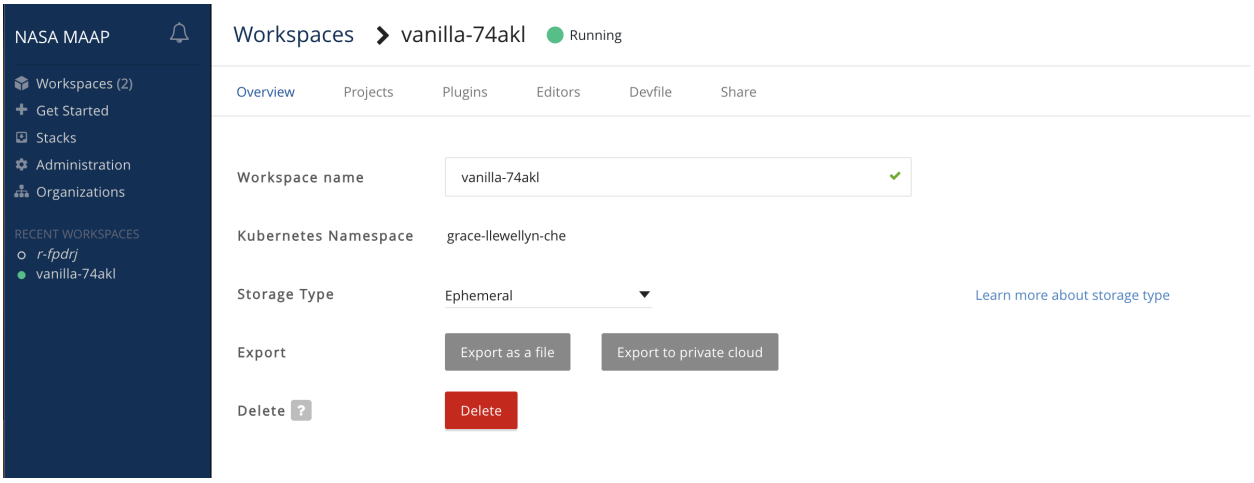


Option 2: Edit Existing Workspace

1. In the workspaces tab under the Che side panel, select the workspace you want to rename.



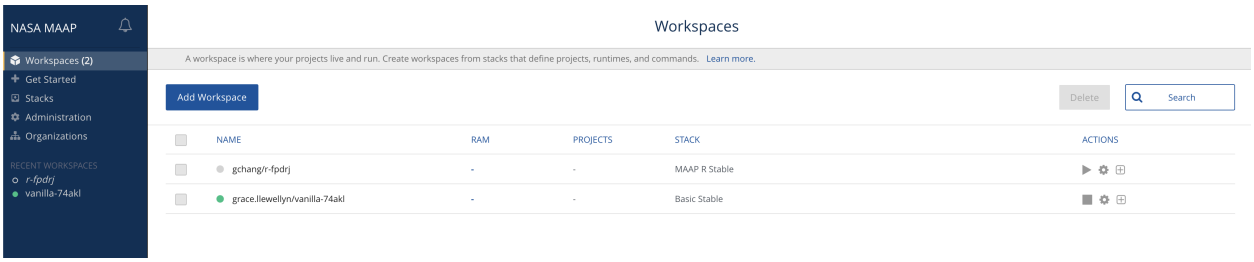
2. Under the Overview section, you can replace the Workspace name field with whatever you wish to name it.



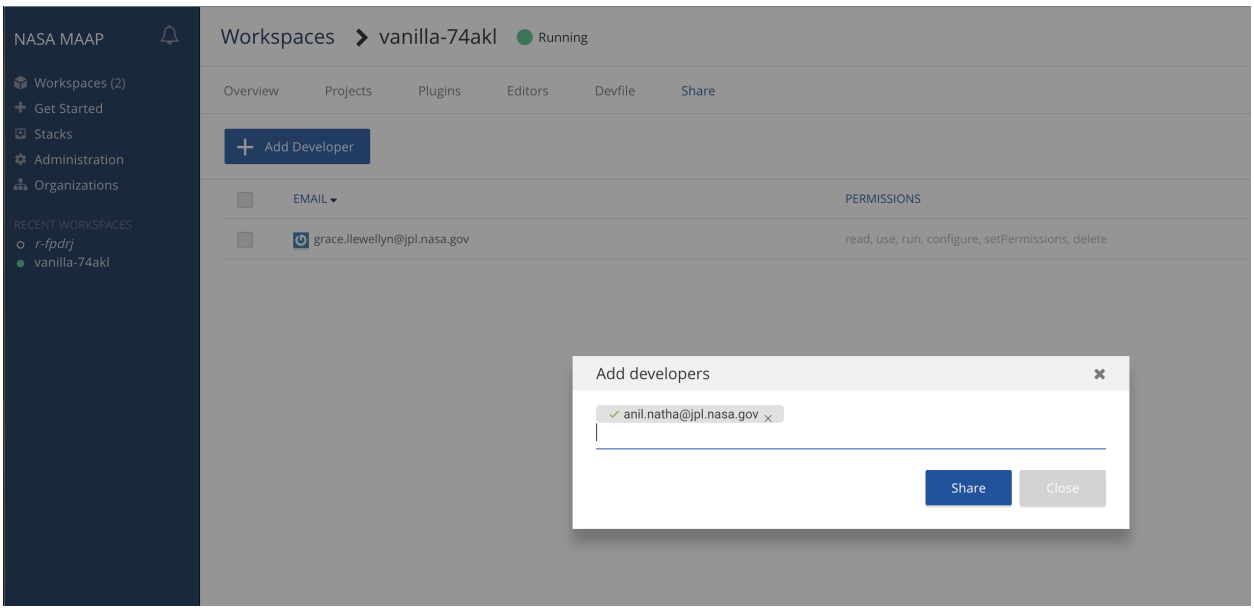
Caveat: no special characters like space, dollar sign, etc; name should start/end only with digits, Latin letters, or underscores

4.1.4 Share My Workspace With Another MAAP User

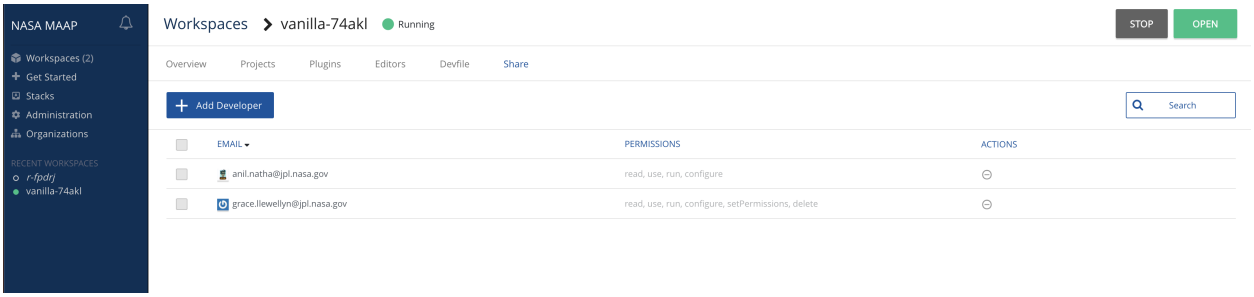
1. In the workspaces tab under the Che side panel, select the workspace you want to share.



2. Under the Share section, click the button + Add Developer. Then type in the email of the person/people you wish to share the workspace with and click *Share*. Only emails registered with MAAP can be used to share workspaces.

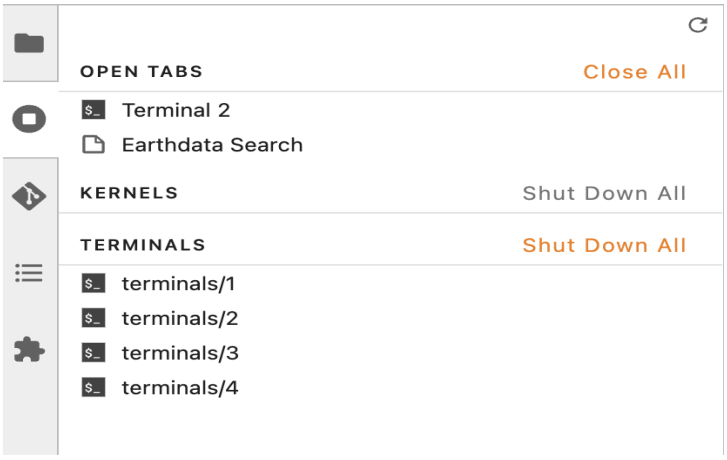


3. The user will now be listed for this workspace.



Important notes:

- If the workspace was opened under an organization, the workspace can only be shared with members of that organization and not any other MAAP user.
- Currently, we do not have a way to denote shared workspaces in the Che side panel, so we recommend renaming your workspace so that you know it is shared (e.g. add _s)
- Users who you share a workspace with will be able to see everything in your `my-private-bucket` folder and will have read, write, and delete permissions
- You will be able to see what tabs other users have open by refreshing or clicking on the "Running Terminals and Kernels" side tab

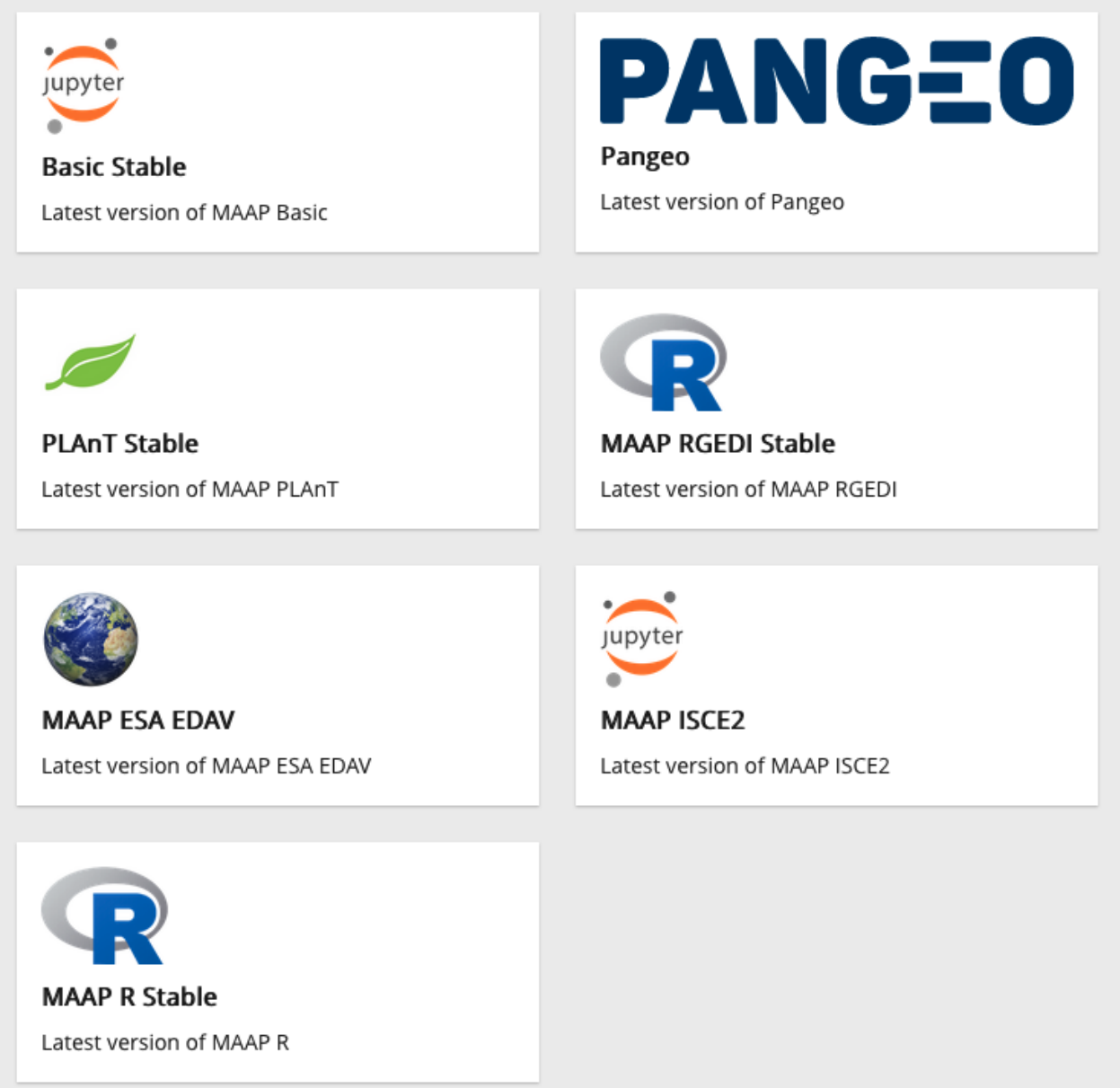


4.2 Environments

This document guides MAAP users in the process of selecting, extending existing environments (the set of libraries availables for analysis) or creating custom environments.

4.2.1 Workspaces

The MAAP ADE offers various workspace options, each workspace coming with its own environment that has pre-installed essential libraries for computing and geospatial analysis. At the time of writing this guide, here are the options :



For example, the `MAAP RGEDI Stable` and `MAAP R Stable` workspace options come with various pre-installed R packages.

For more information: Each of these options rely on Docker images that were build off from Dockerfiles that are

publicly available in the MAAP workspace repository. If you want to learn more about what libraries each image contains, check out this repository.

4.2.2 Extending environments

Users may need libraries for their specific analysis purposes that are not present in the environments of the different workspace options offered. In this case, ideally, the steps should be the following :

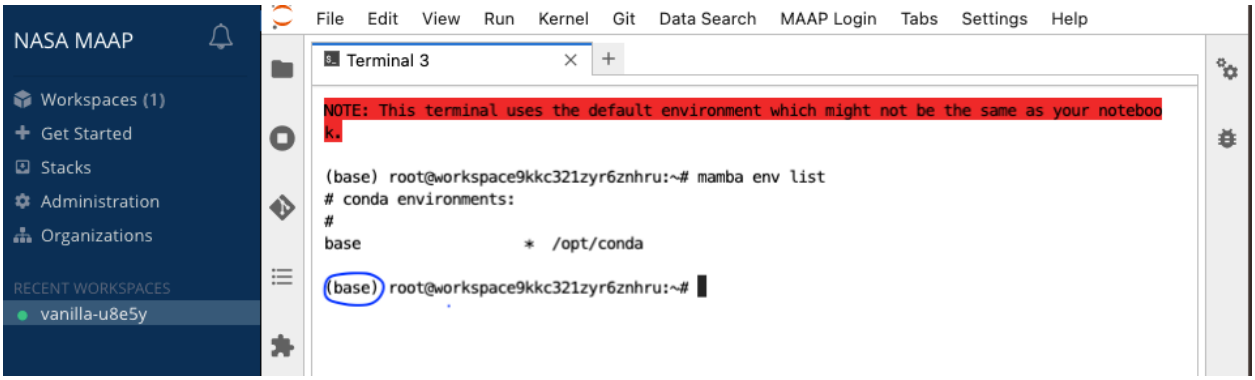
1. The user explores her/his environment need by extending the environment of an existing workspace or creating her/his custom environment in an existing workspace (see next sections).
2. Once that is done, the user submits a ticket/coordinates with the platform team to create a new workspace option with the requested, finalized environment.

The above approach is ideal because modifications to the pre-defined workspace environment do not survive a workspace restart (see next sections), and because sharing new experimented environments is valuable.

The next sections explain how to extend environments or create custom environments, and for this, introduces information regarding which environment management solution we are using.

Package manager

We use `conda` with the libmamba solver as a package manager to install, update or remove packages (libraries). `conda` works with 'environments' that are directories in your local file system containing a set of packages. When you work 'in a given environment', it means that your programs will look for dependencies in that environment's `conda` directory. All workspaces launch with an environment called `base`, which is a `conda` environment that has all the pre-installed libraries. If you open a terminal launcher after creating a `Basic Stable` workspace :



You can notice that a `base` `conda` environment is activated, and its libraries are located in `/opt/conda`.

Extending the `base` environment in a given workspace session.

Note : any modification to the ``base`` environment does not survive a workspace restart. In other words, modifications to ``/opt/conda`` disappear after a workspace restart.

Extending an existing `conda` environment means adding packages on top of what it contains, which works provided there are no dependency conflicts. You can install libraries using the `conda install` command to install additional packages in your current environment (run `conda --help` to learn more about how to use `conda` commands). All `conda` install commands should use `-c conda-forge` otherwise it's unlikely to work, since many/most of the packages installed already are from conda-forge. For example :

```
conda install -c conda-forge xarray
```

libmamba is the default solver, but users are welcome to set the solver to "classic" with:

```
conda install --solver=classic -c conda-forge xarray
```

However, it is recommended to use configuration files for reproducibility and shareability. With this approach, assuming your configuration file is named `config.yml`, the command to use is :

```
conda env update -f config.yml
```

For more details on configuration files, see the *Custom environments section* and for an example of this command, refer to the *subsection about updating an environment with a configuration file*.

4.2.3 Custom environments

For the rest of this README, in each section we provide a link to download an example YAML configuration file.

You can use the `conda` CLI to create a new, custom environment. The parameters (the list of libraries, the location where to search for them, etc…) can be passed either from a configuration YAML file or directly on the console. We recommend using the first option (a YAML file is easier to share and modify).

Basic custom environment

*Example config file for this section*here.

This configuration installs specific versions `python`, `pandas` and `geopandas` from `conda-forge`. If versions aren't specified, the latest is installed. We recommend to always specify the version for reproducibility. The basic command to create this environment would be :

```
conda env create -f env-example.yml
```

However, this stores this environment files in `/opt/conda`, which is a directory that is recreated when the workspace restarts, and so custom environments are lost. Therefore, you want to specify a storage location in your user directory with the `--prefix` parameter

```
conda env create -f env-example.yml --prefix /projects/env/env-example
```

and to activate it :

```
conda activate /projects/env/env-example
```

Updating an existing environment with a configuration file

*Example config file for this section*here.

You can *update* an existing environment with a configuration file as well. For example, let's assume you have a `conda` environment with a set of packages already installed in it (for example the `base` environment), but it doesn't have `xarray` and `geopandas`. Using the linked example config :

```
conda env update -f env-extend.yml
```

This command will update the active environment by adding `xarray` and `geopandas`, provided it does not cause conflicts with the existing libraries.

Using `pip` for python packages

*Example config file for this section*here.

Some python packages might not be availabe in the channel you are using, or in any `conda` channel. If that package however is in `PyPI` (the official python package repository), one can use `pip` within a `conda` environment to download packages. The recommended way is to specify this in the configuration file. In the linked example, we add `stackstac` as a dependency to install from `PyPI` because it is not available in the `conda-forge` channel.

Using custom environments in jupyter notebooks

*Example config file for this section*here.

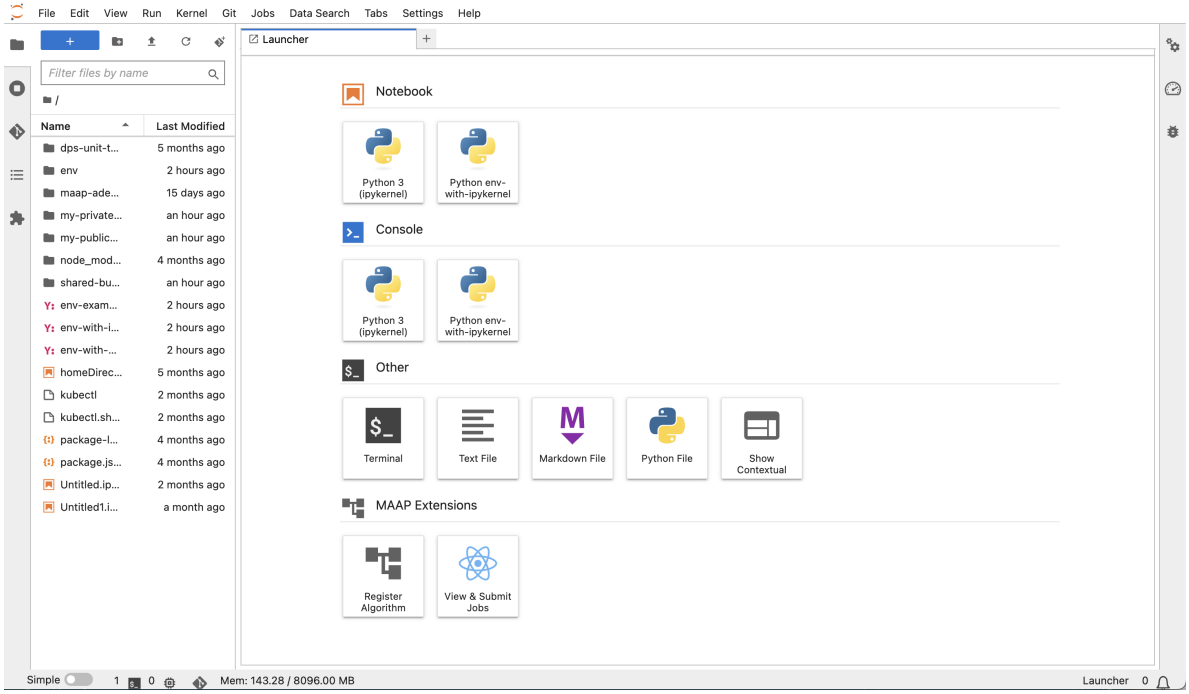
The following instruction steps are for python kernels.

- Make sure ipykernel is listed as a dependency in your configuration file.
- Create your environment using the linked configuration file.
- Install the environment as a kernel by running the following command (parameter values follow the example mentioned):

```
python -m ipykernel install --user --name env-with-ipykernel --display-name "Python␣
↪env-with-ipykernel"
```

  The above command installs the environment as a kernel in Jupyter, making it accessible in the notebook with a display name of "Python env-with-ipykernel".

- Wait around 30 seconds and launch a new notebook. Among the kernel options, you should see "Python env-with-ipykernel" listed. Below you can see a screenshot that shows what this step looks like:



- Remove by listing kernelspecs `jupyter kernelspec list` to find name, then `jupyter kernelspec remove <env>`

Suggested packages for custom environment

*Example config file for this section*here

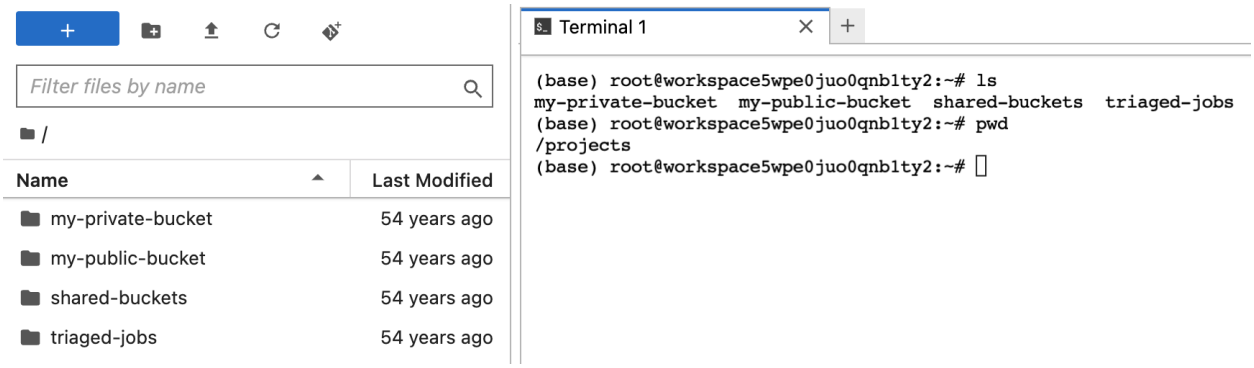
MAAP users typically use the python `maap-py`. It's pre-installed in all workspaces, in the `base` conda environment, but any custom environment should specify it, otherwise it is not going to be accessible from that environment. However, `maap-py` is not packaged in a public package repository, like `PyPI` or `conda-forge`. It is possible to install it directly from its github repository with `pip` though. See the configuration example linked. You can note that in the example, `maap-py` is 'versioned' using a commit hash (at the end of the github URL).

4.3 Share Data

Users who have access to a workspace have access to all the files contained in that workspace.

All users have their own personal s3-bucket folder mounted in `/projects`, called `my-private-bucket`. Each user has a public s3-hosted folder called `my-public-bucket`. Files in this folder are automatically uploaded to s3 and are accessible from any workspace a user signs into.

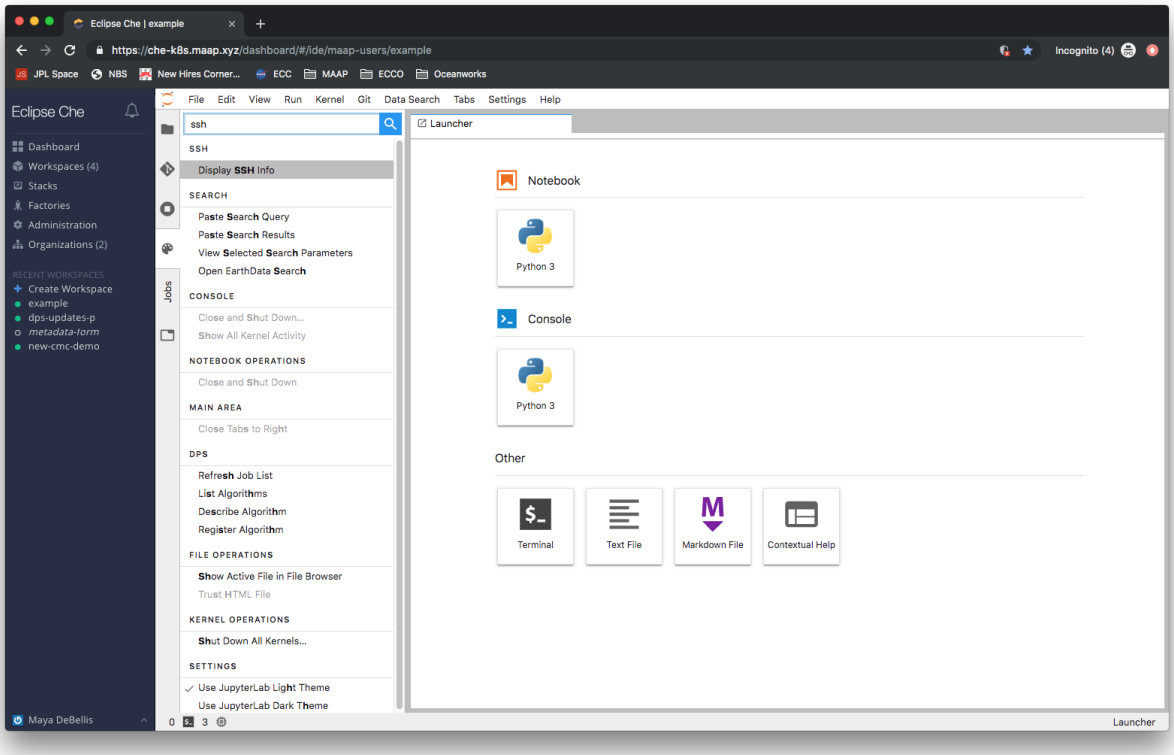
The intention of this mounted folder is that you can use this to share data with others, and also store files you want to access across different workspaces. It is not intended that you do all of your work in this directory. Because this directory is mounted to s3, you will notice that processes are slower when working in this directory. Other users can access what you put in your public bucket by going to `shared-buckets` and the folder marked with your CAS username.

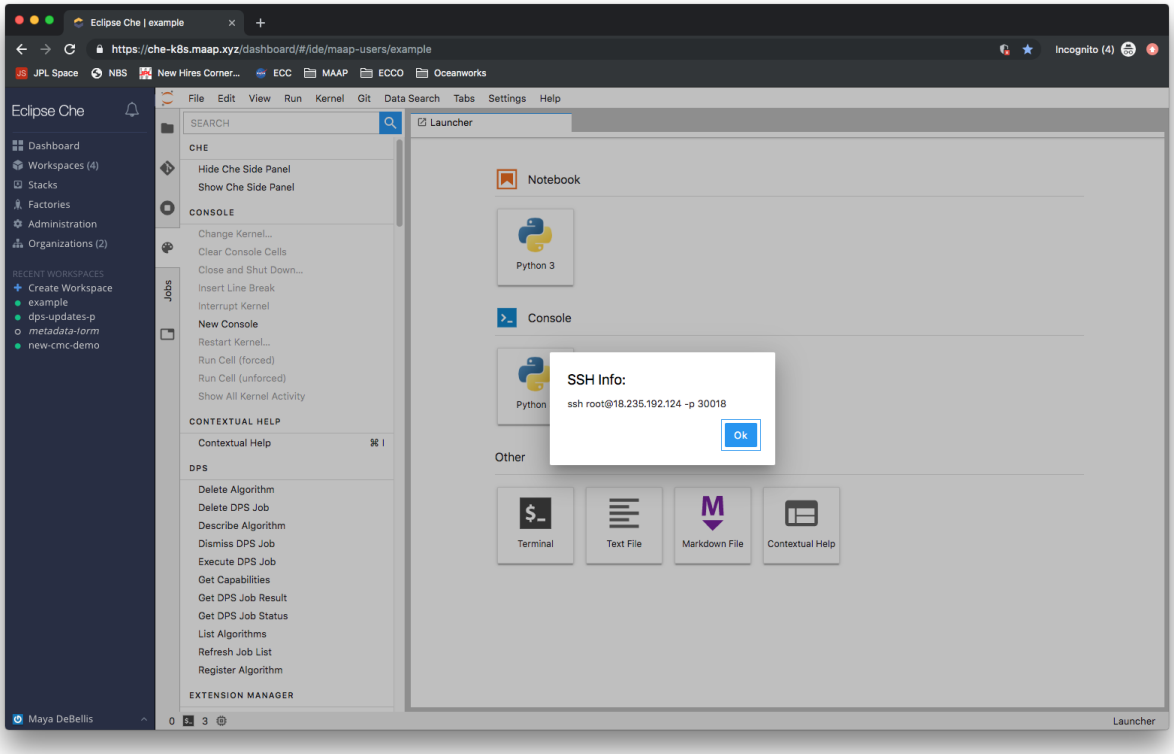


Users can create a shareable link for any files in their folder that is hosted on s3. To do this, go to `Command Palette -> User -> Get Presigned S3 Url` and enter the relative path to the file you want to share. You can also do this by right clicking a file and selecting "Get Presigned S3 Url". The link will expire after 12 hours.

4.4 SSH into Workspace

As an alternative to using the jupyter interface, you can SSH directly into the container that your workspace set up. In order to get the IP and port information, navigate to the command palette of the jupyter interface. Find the command `Display SSH Info`, which will display the information you need (you can easily search for SSH). Your public SSH key that you added to your account will be added to any workspace you create. If you did not upload an SSH key to your profile, you will not be able to SSH in and must use the jupyter interface.


Eclipse Che | example
https://che-k8s.maap.xyz/dashboard/#/ide/maap-users/example
Incognito (4)
JPL Space
NBS
New Hires Corner...
ECC
MAAP
ECCO
Oceanworks
Eclipse Che
Dashboard
Workspaces (4)
Stacks
Factories
Administration
Organizations (2)
RECENT WORKSPACES
Create Workspace
example
dps-updates-p
metadata-torm
new-cmc-demo
Maya DeBellis
File Edit View Run Kernel Git Data Search Tabs Settings Help
ssh
SSH
Display SSH Info
SEARCH
Paste Search Query
Paste Search Results
View Selected Search Parameters
Open EarthData Search
CONSOLE
Close and Shut Down...
Show All Kernel Activity
NOTEBOOK OPERATIONS
Close and Shut Down
MAIN AREA
Close Tabs to Right
DPS
Refresh Job List
List Algorithms
Describe Algorithm
Register Algorithm
FILE OPERATIONS
Show Active File in File Browser
Trust HTML File
KERNEL OPERATIONS
Shut Down All Kernels...
SETTINGS
Use JupyterLab Light Theme
Use JupyterLab Dark Theme
Launcher
Notebook
Python 3
Console
Python 3
Other
Terminal
Text File
Markdown File
Contextual Help
Launcher


Eclipse Che | example
https://che-k8s.maap.xyz/dashboard/#/ide/maap-users/example
Incognito (4)
JPL Space
NBS
New Hires Corner...
ECC
MAAP
ECCO
Oceanworks
Eclipse Che
Dashboard
Workspaces (4)
Stacks
Factories
Administration
Organizations (2)
RECENT WORKSPACES
Create Workspace
example
dps-updates-p
metadata-torm
new-cmc-demo
Maya DeBellis
File Edit View Run Kernel Git Data Search Tabs Settings Help
SEARCH
CHE
Hide Che Side Panel
Show Che Side Panel
CONSOLE
Change Kernel...
Clear Console Cells
Close and Shut Down...
Insert Line Break
Interrupt Kernel
New Console
Restart Kernel...
Run Cell (forced)
Run Cell (unforced)
Show All Kernel Activity
CONTEXTUAL HELP
Contextual Help ⌘ I
DPS
Delete Algorithm
Delete DPS Job
Describe Algorithm
Dismiss DPS Job
Execute DPS Job
Get Capabilities
Get DPS Job Result
Get DPS Job Status
List Algorithms
Refresh Job List
Register Algorithm
EXTENSION MANAGER
Launcher
Notebook
Python 3
Console
Python
SSH Info:
ssh root@18.235.192.124 -p 30018
Ok
Other
Terminal
Text File
Markdown File
Contextual Help
Launcher

4.4.1 Accessing MAAP workspaces over SSH

If you would like to have your MAAP workspace mounted (via SSH) on your local computer, follow these steps. It is completely optional and if you do not know why you would want this, feel free to skip this part of the Getting Started Guide.

Basically, this works the same as on any unix-based system: You need to add your Public SSH key from your computer/laptop to the `~/.ssh/authorized_keys` file in MAAP, as described below.

Generate a Public SSH Key

First, you need an SSH key for your personal machine. Here are example instructions for creating an SSH key. Note that there are different instructions for Windows and unix-based systems (MacOS, Linux). The outcome of this will be a Public SSH key on your local computer.

Add the Public SSH Key to MAAP

By default, you cannot see or edit hidden files in the MAAP Jupyter account. Therefore, you should do the following workaround:

1. On your local computer, copy the contents of the public SSH key you just created. It will usually have a `.pub` extension (the default file is `~/.ssh/id_rsa.pub`) and its contents will look something like this (one long line):

   ```
   ssh-rsa AAAAB3Nza[...]TeDx1 ashiklom@gs610-ashiklom.ndc.nasa.gov
   ```

2. Use the Jupyter interface (right click in the file browser) to create a new folder in your home directory called `ssh`.
3. Inside that folder, use the Jupyter interface (right click in the file browser) to create a new file called `authorized_keys` (NOTE: no file extension!).
4. Double click the file you just created to open it in the editor.
5. Copy the contents of your SSH key from your computer into this file. Save it and close it.
6. Return to your home directory, right click the `ssh` folder, click "Rename", and rename it to `.ssh` (period in front). You will get a 404 error, but the operation will have succeeded, and folder will 'disappear'.
   - To check that this worked, in a terminal window, run `ls -a` in your home directory. You should see the `.ssh` folder in there.
7. Repeat these steps to add more SSH keys from different machines.
   - Note that, to see the `.ssh` folder from JupyterLab, you'll need to temporarily rename it to `ssh`. The only way to do this is via the terminal — `mv .ssh ssh`. Don't forget to rename it back to `.ssh` when you're done!

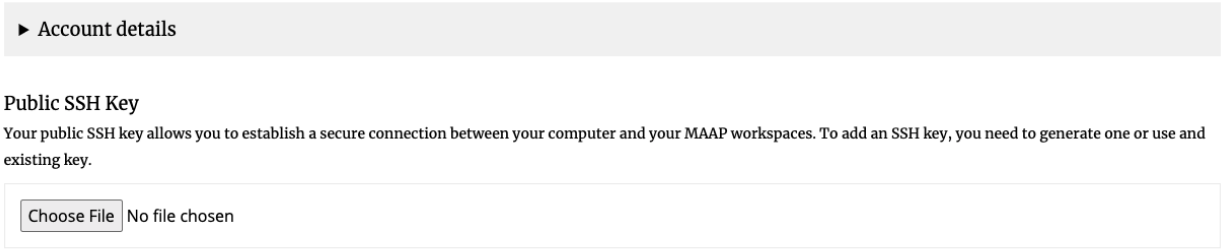
To access a MAAP workspace over SSH, you will need a workspace-specific IP address and port. You can find this by going to the JupyterLab commands menu — the 4th button in the left panel of the Jupyter interface; shows a looking glass over a list; alternatively, press Control + Shift + C (Command + Shift + C on Mac) — and search for "SSH". You'll see "Display SSH Info". Click this button. A dialog box with the correct SSH command will appear.

Uploading your public SSH key

In order to access your ADE workspaces using SSH, you'll need to upload your public SSH key to your MAAP profile using the MAAP portal at https://maap-project.org.

Click on your profile name (or the "Login" button) in the top right corner shown here:

On your profile page, click the "Choose File" button to upload your key.



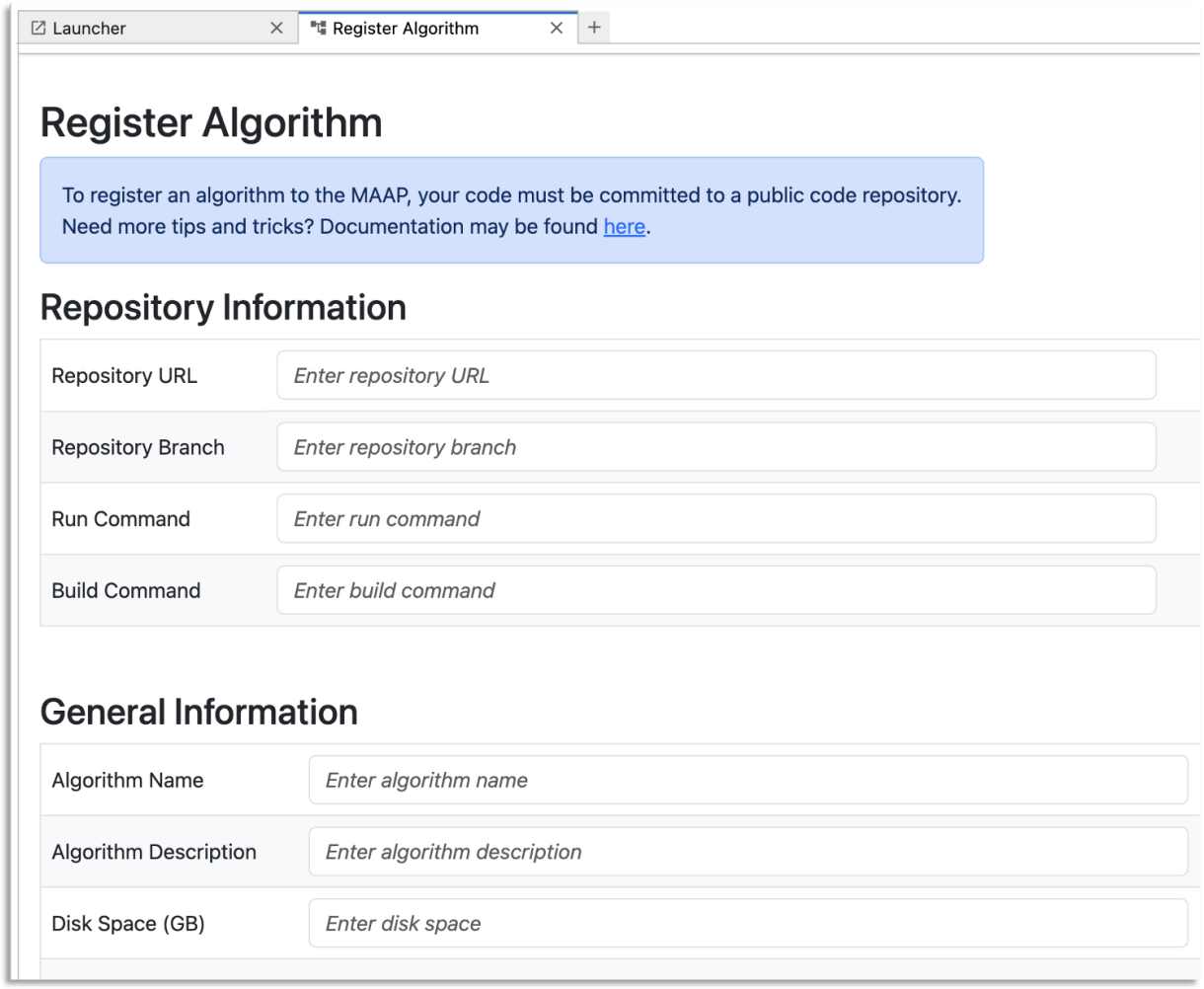
After uploading your key, the SSH connection will be enabled **after your next** login into the ADE.

4.5 Algorithm Registration UI

The algorithm registration UI allows users to register algorithms stored in their public code repositories to the MAAP so the algorithms can then be used to run jobs on the MAAP.

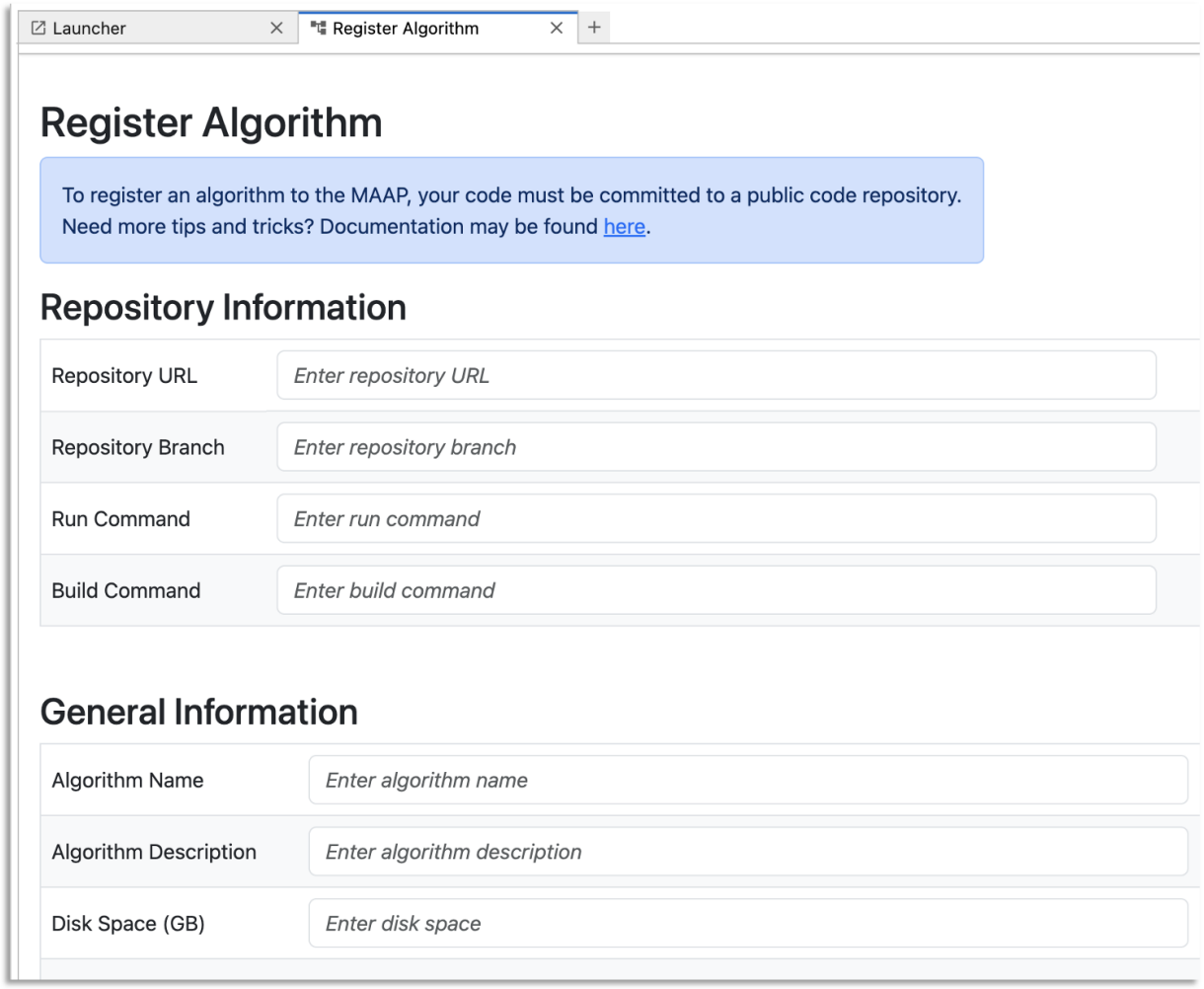
4.5.1 Access

1. Open your workspace
2. Click the **Register Algorithm** button from the launcher menu.

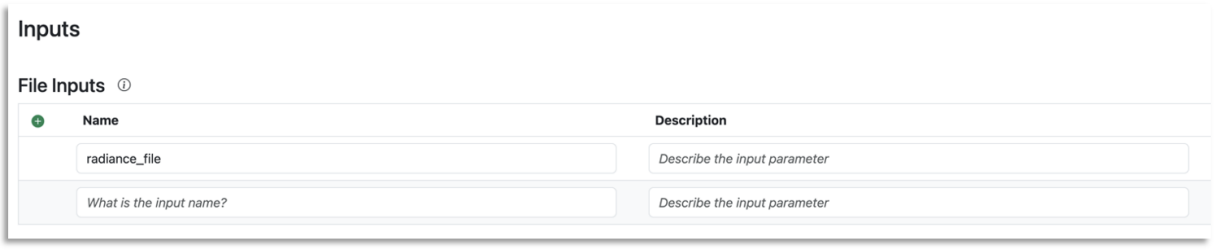


4.5.2 Register an Algorithm

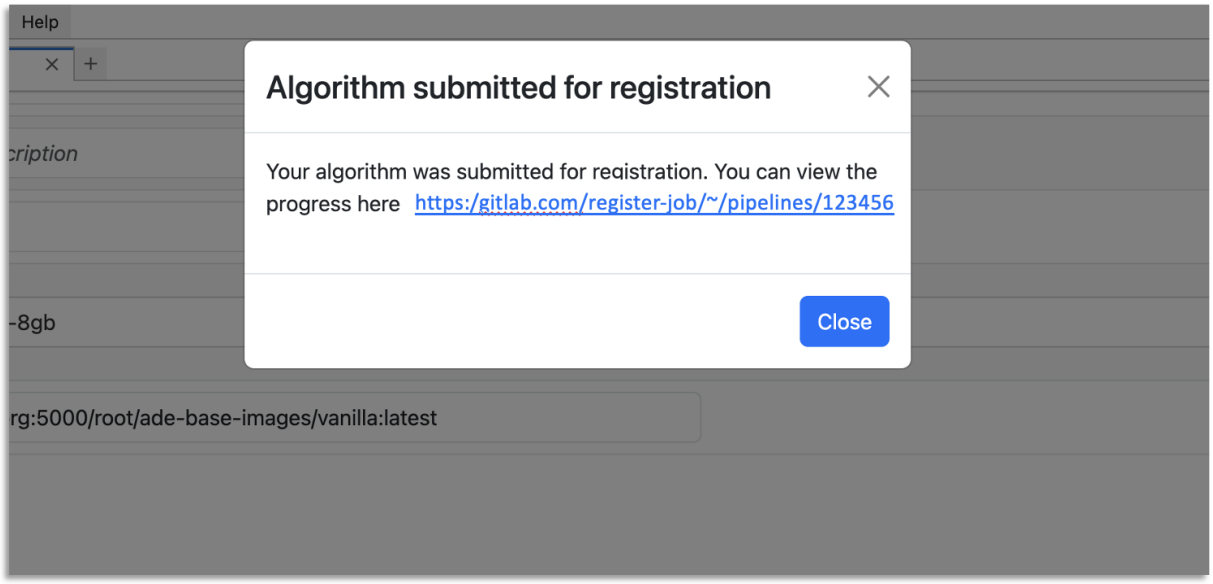
1. Open the **Register Algorithm** UI.
2. Fill out the fields describing general information about the algorithm. This includes things like the repository the algorithm is stored in, the repository branch (also refered to as the version of the algorithm), the command to execute the algorithm, etc.



3. Fill out the algorithm inputs. To add a new input, click the green plus circle. To remove an input click the red x circle.



4. Once all information is filled out, click the **Register Algorithm** button at the bottom of the form. After a few seconds, a modal will appear with the link to the registration job that was just submitted. Users can navigate to this link to access the registration logs and confirm their algorithm was registered successfully.

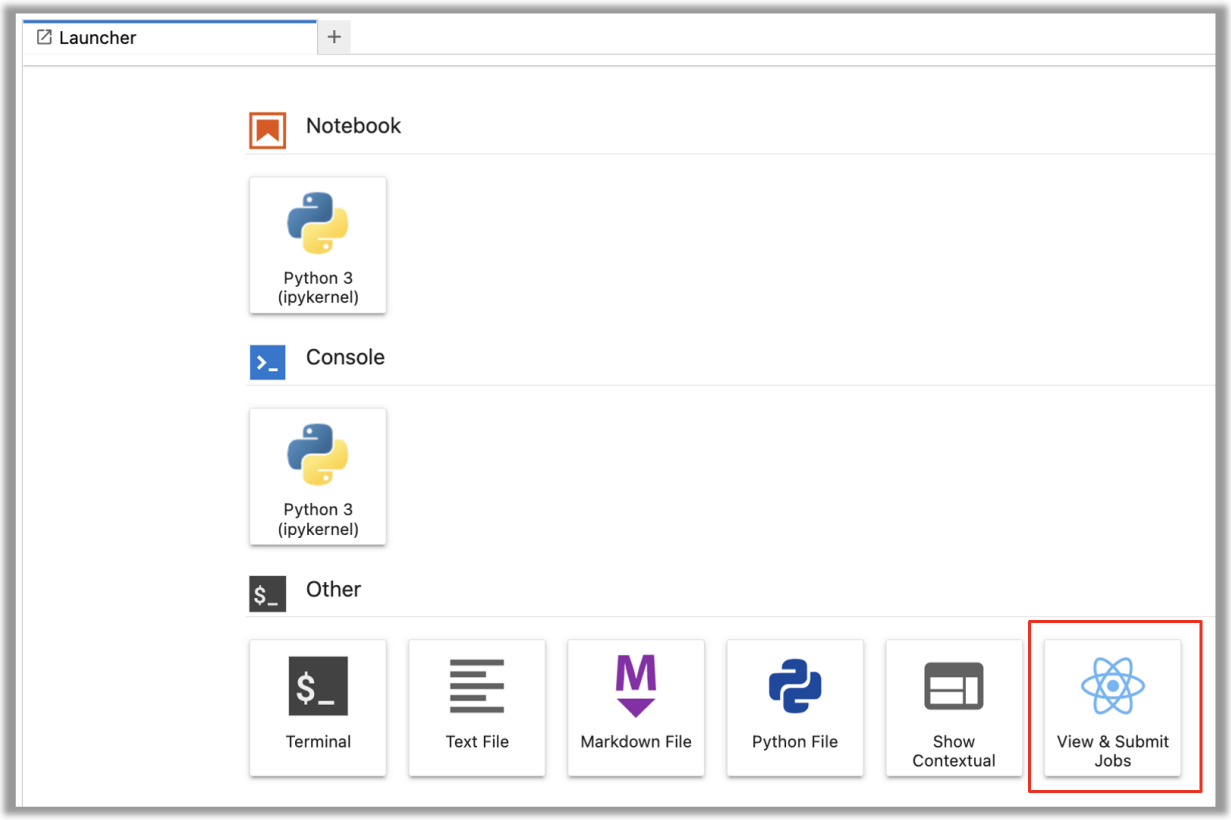


4.6 Jobs UI for Job Management

The Jobs UI allows users to submit and view DPS jobs from their Jupyter workspace. Users can monitor job status, access generated products, view errors, and view other job metadata.

4.6.1 Access the Jobs UI

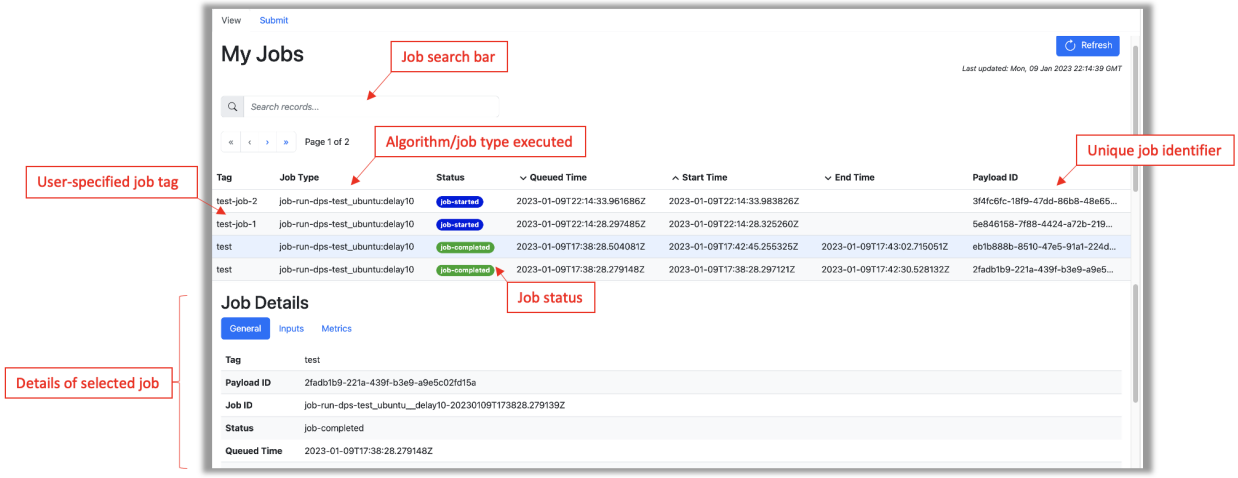
1. From your workspace click on the **View & Submit Jobs** card on the Launcher tab.



4.6.2 View Jobs

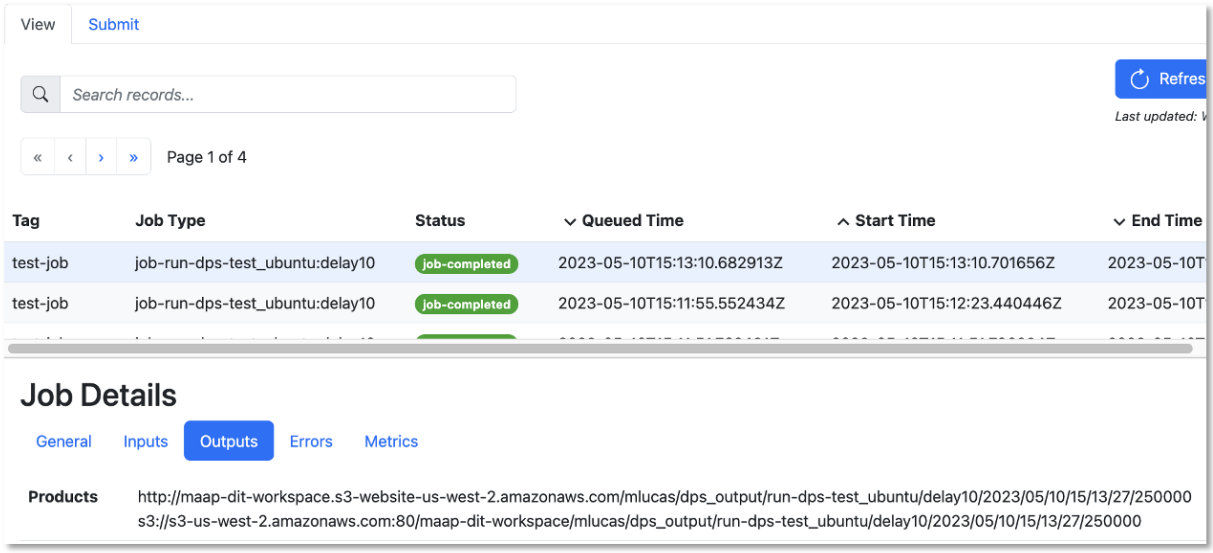
The **View** pane lists all the jobs submitted by the user. The top table shows only a few key fields. Users can click on any row to view detailed data for the selected job such as inputs, outputs generated, and errors produced if the job failed to complete successfully.

Users can sort jobs by queued, start, and end time in ascending/descending order. Users may use the search bar to filter the job list down to jobs containing the user-provided string in any of the fields shown.



4.6.3 Viewing the Output of a Job

Select your job then under the `Job Details` table, click the `Outputs` tab. The `Products` field provides the path to the product directory within the workspace.



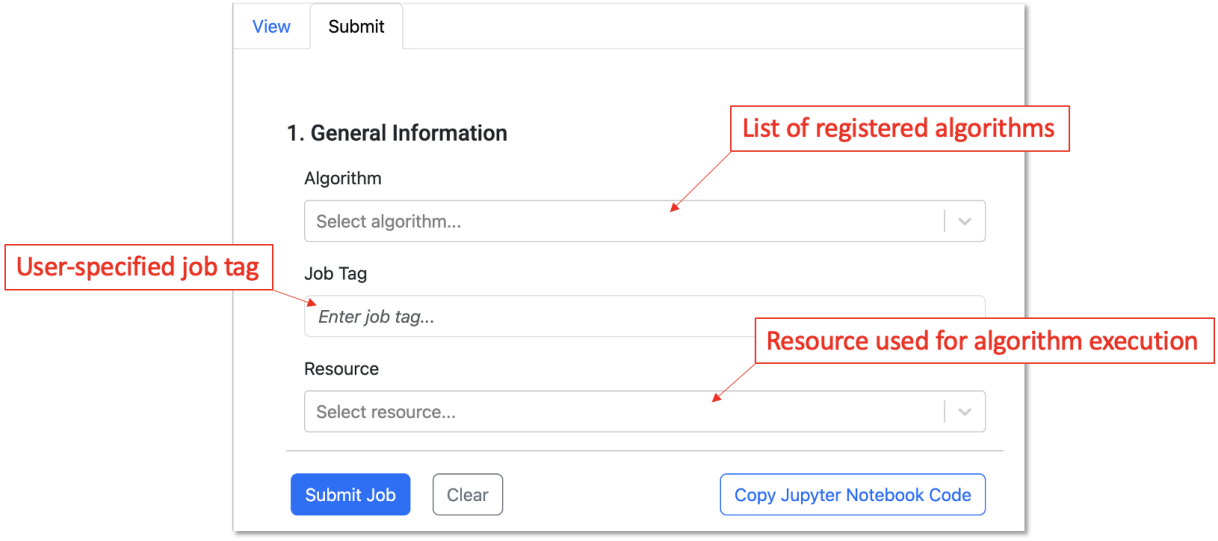
Navigating to the product directory for the selected job above shows the following:

```
(base) root@workspacelbx1uqgeyqdei83w:~/my-private-bucket/dps_output/run-dps-test_ubuntu/
↪delay10/2023/05/10/15/13/27/250000# pwd
/projects/my-private-bucket/dps_output/run-dps-test_ubuntu/delay10/2023/05/10/15/13/27/
↪250000

(base) root@workspacelbx1uqgeyqdei83w:~/my-private-bucket/dps_output/run-dps-test_ubuntu/
↪delay10/2023/05/10/15/13/27/250000# ls
_stderr.txt  _stdout.txt  output-2023-05-10T15:13:27.250000.context.json  output-2023-05-
↪10T15:13:27.250000.dataset.json  output-2023-05-10T15:13:27.250000.met.json  write-
↪output.txt
```

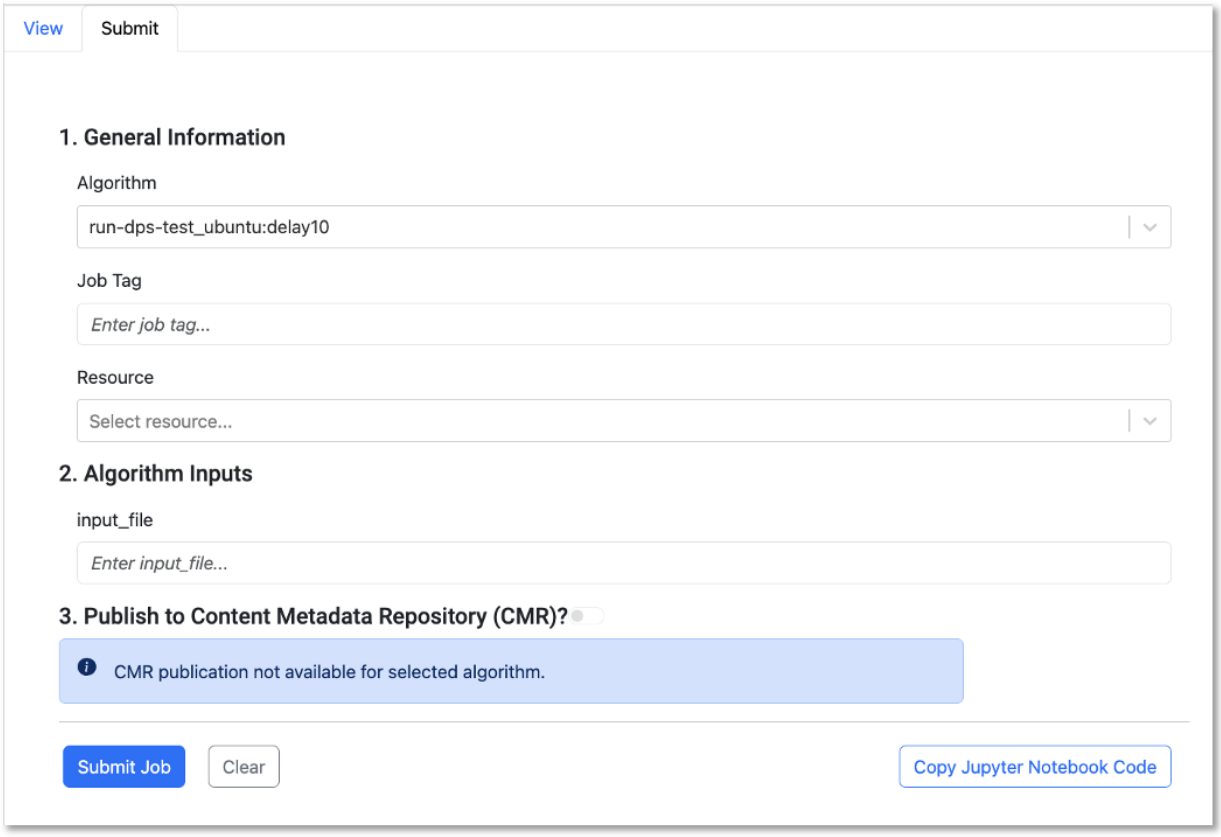
4.6.4 Submit a Job

Users can submit individual jobs from the **Submit** pane. The following are the minimum requirements for submitting a job:

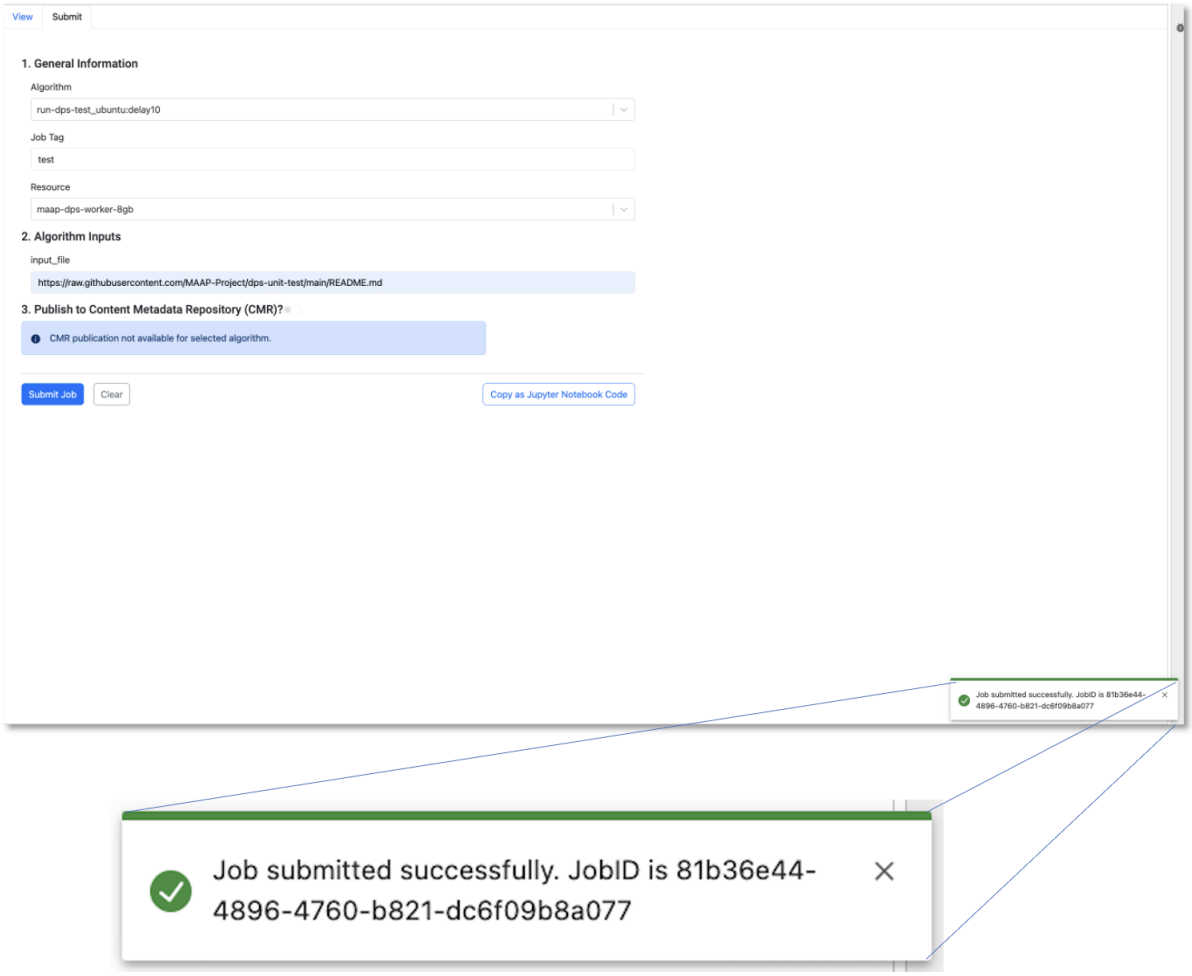
1. Select an algorithm from the dropdown menu. Note: only registered algorithms will be shown.
2. Provide a tag that may then be used to easily search for and identify the submitted job.
3. Select the resource to use for algorithm execution.



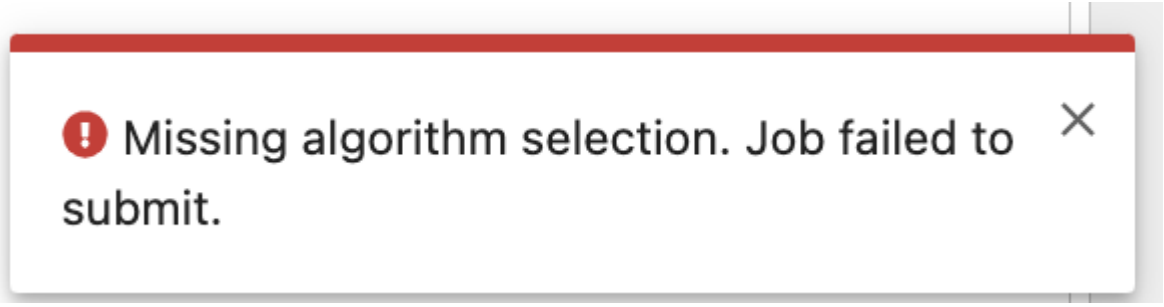
Algorithms may contain additional inputs that users may have to provide.



Once all inputs have been provided, the user may click **Submit Job** to submit the job. If the job was submitted successfully, a toast will appear in the bottom right corner containing the unique job id. Please give the **Submit Job** button a second to execute.

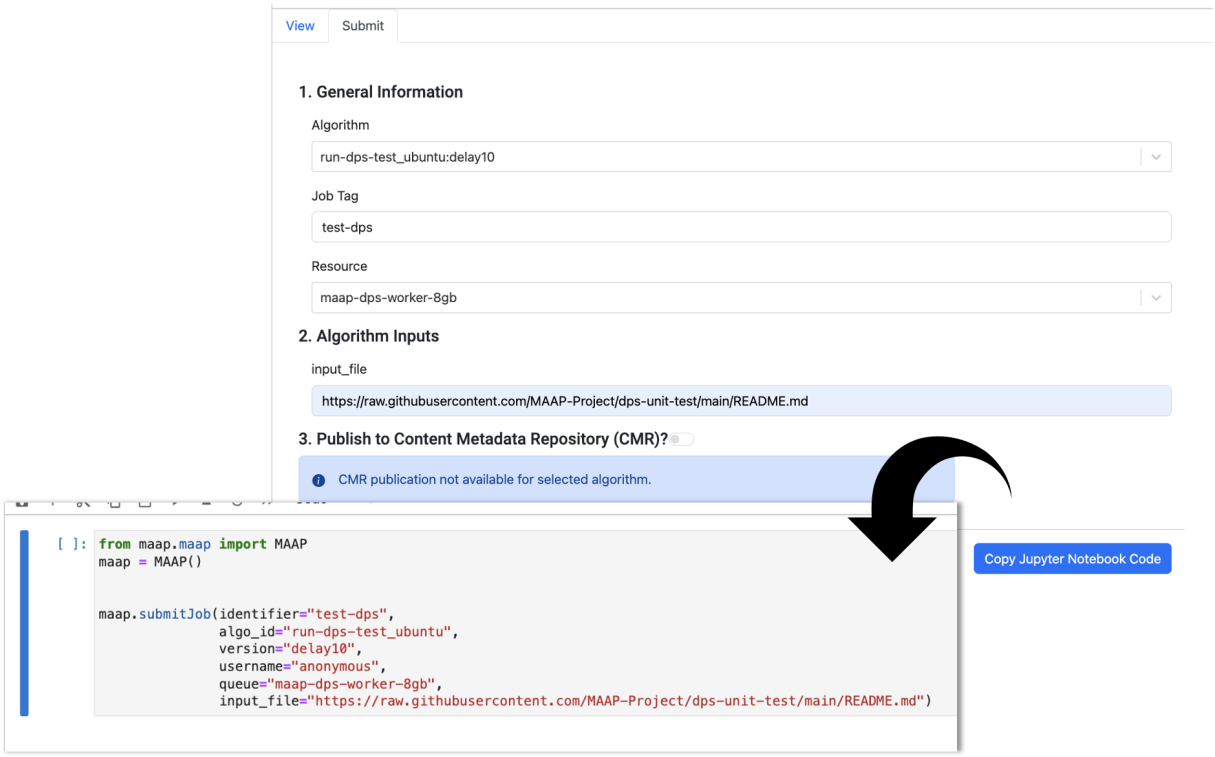


If the job failed to submit, a toast will appear indicating the job failed to submit.



4.6.5 Generate Job Submission Command

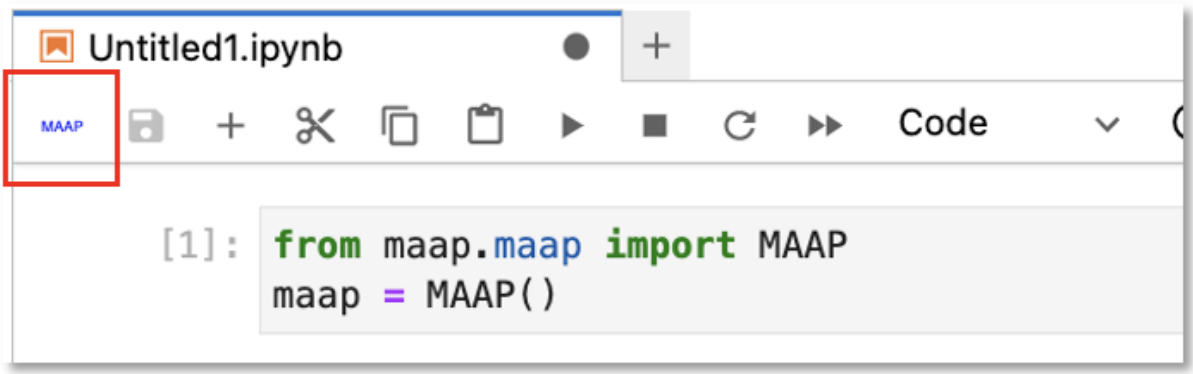
Users may fill out the job submission form and - instead of submitting the job - click the **Copy Jupyter Notebook Code** button to copy the **maap-py** job submission command to their clipboard to then paste it into a Jupyter notebook.



4.7 Job Management with maap-py

4.7.1 Set up maap.py

1. Open a Jupyter Notebook then click the `MAAP` button from the notebook toolbar. This will paste the code snippet below into your notebook.



2. Provide the MAAP host. For normal operations, this would be `api.maap-project.org`.

```
from maap.maap import MAAP
maap = MAAP(maap_host='api.maap-project.org')
```

4.7.2 Submit a Job

Use the `submitJob` method and provide your algorithm inputs. The example below will run the `run-dps-test_ubuntu` algorithm.

```
maap.submitJob(identifier="test-job",
               algo_id="run-dps-test_ubuntu",
               version="delay10",
               username="anonymous",
               queue="maap-dps-worker-8gb",
               input_file="https://raw.githubusercontent.com/MAAP-Project/dps-unit-test/
↪main/README.md")
```

4. Run the notebook to submit the job. The cell output for a job that was submitted successfully will look similar to this:

```
{'status': 'success',
 'http_status_code': 200,
 'job_id': '86fbac52-24b0-4963-8b67-59d0fc09946a'}
```

4.7.3 Monitor a Job

1. Use the `getJobStatus` method and provide the job ID that was created upon job submission.

```
r = maap.getJobStatus("86fbac52-24b0-4963-8b67-59d0fc09946a")
r.text
```

2. Run the notebook to get the job status. The output should resemble the xml snippet below. In this example, the job status is `Succeeded`.

```
'<wps:StatusInfo xmlns:ows="http://www.opengis.net/ows/2.0" xmlns:schemaLocation="http://
↪schemas.opengis.net/wps/2.0/wps.xsd" xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:
↪xsi="http://www.w3.org/2001/XMLSchema-instance"><wps:JobID>86fbac52-24b0-4963-8b67-
↪59d0fc09946a</wps:JobID><wps:Status>Succeeded</wps:Status></wps:StatusInfo>'
```

4.7.4 Get Job Results

1. Use the `getJobResult` method and provide the job ID that was created upon job submission.

```
r = maap.getJobResult("86fbac52-24b0-4963-8b67-59d0fc09946a")
r.text
```

2. Run the notebook to get the job result. The output should resemble the xml snippet below.

```
<wps:Result xmlns:ows="http://www.opengis.net/ows/2.0" xmlns:schemaLocation="http://
↪schemas.opengis.net/wps/2.0/wps.xsd" xmlns:wps="http://www.opengis.net/wps/2.0" xmlns:
↪xsi="http://www.w3.org/2001/XMLSchema-instance"><wps:JobID>86fbac52-24b0-4963-8b67-
```

(continues on next page)

(continued from previous page)

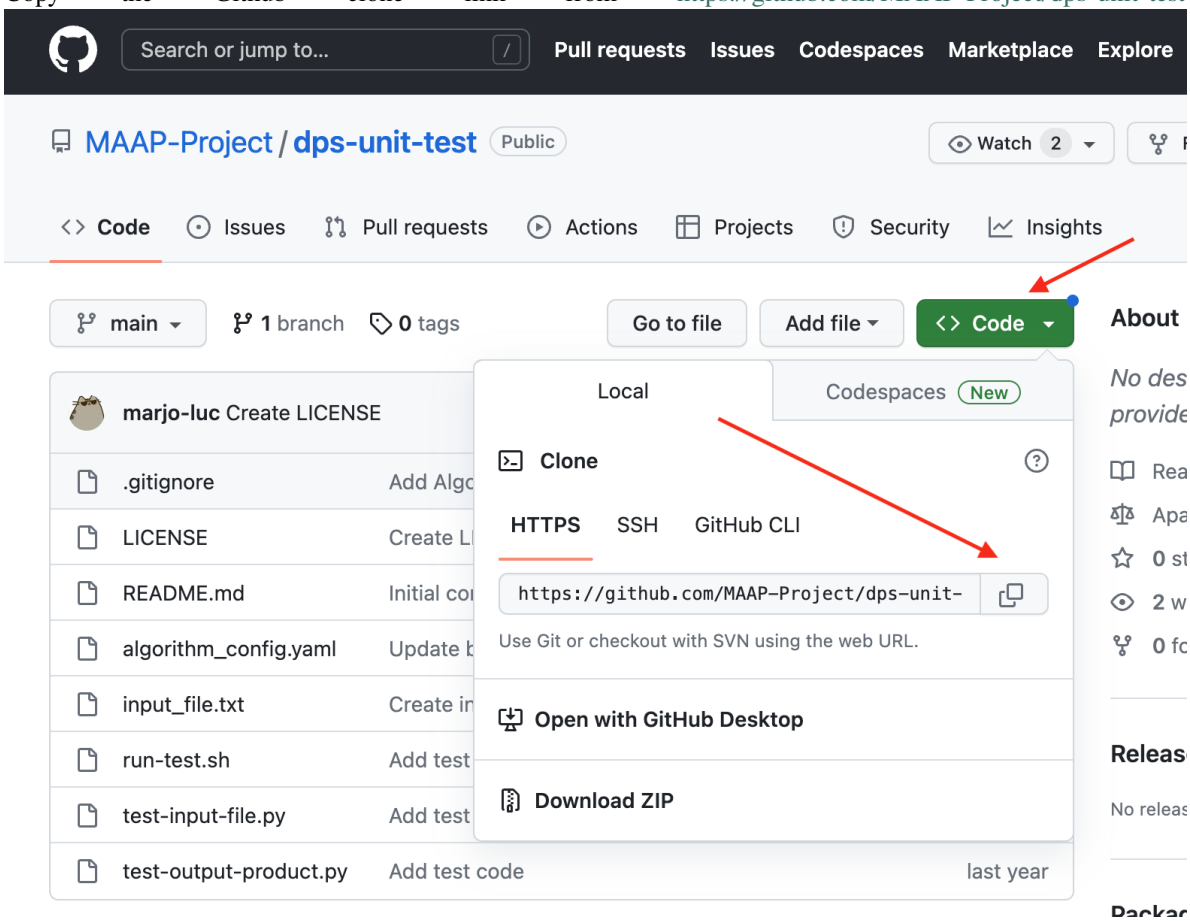
```
↪59d0fc09946a</wps:JobID><wps:Output id="output-2023-05-10T15:39:51.905070"><wps:Data>
↪http://maap-dit-workspace.s3-website-us-west-2.amazonaws.com/anonymous/dps_output/run-
↪dps-test_ubuntu/delay10/2023/05/10/15/39/51/905070</wps:Data><wps:Data>s3://s3-us-west-
↪2.amazonaws.com:80/maap-dit-workspace/anonymous/dps_output/run-dps-test_ubuntu/delay10/
↪2023/05/10/15/39/51/905070</wps:Data><wps:Data>https://s3.console.aws.amazon.com/s3/
↪buckets/maap-dit-workspace/anonymous/dps_output/run-dps-test_ubuntu/delay10/2023/05/10/
↪15/39/51/905070/?region=us-east-1&amp;tab=overview</wps:Data></wps:Output></wps:Result>
```

4.8 Working with Git

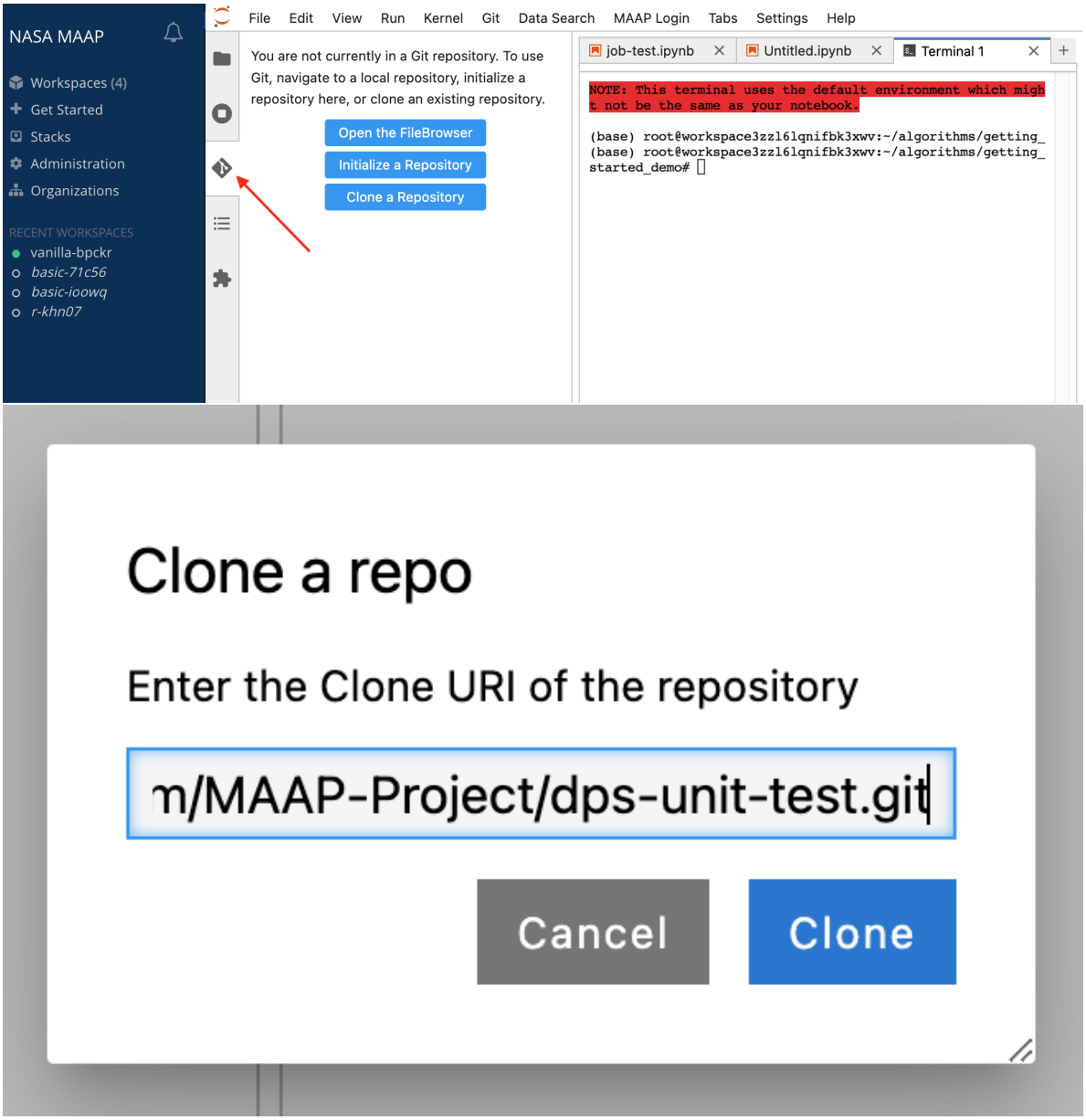
4.8.1 Clone a Repository with GitHub

Here is an example repository you can use for this getting started guide: https://github.com/MAAP-Project/dps-unit-test

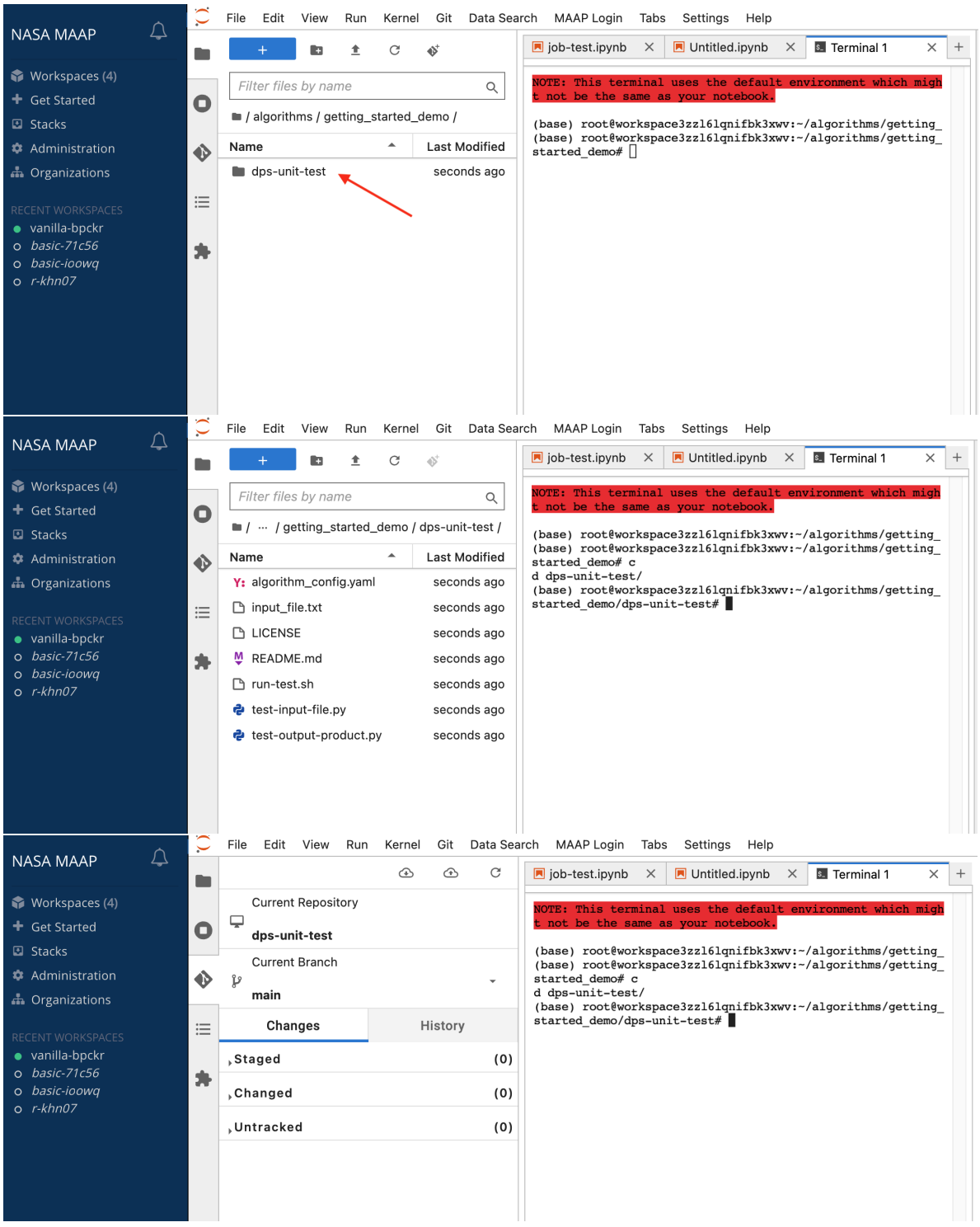
1. Copy the Github clone link from https://github.com/MAAP-Project/dps-unit-test



2. Open the built-in Jupyter Github UI to the left of the file browser. Choose "Clone a Repository" and paste in the .git link you copied from the Github repository. You can also access this menu through the **Git** tab at the top of the Jupyter window.

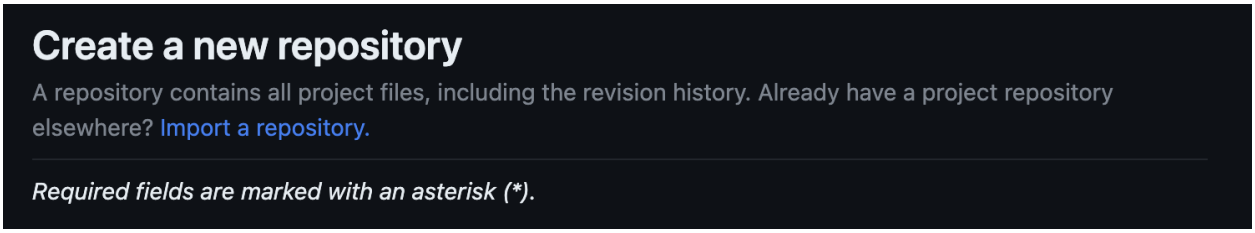


3. You should see a new folder created with the repo you cloned. If you browse to that folder and open up the Jupyter Github UI again, it will show you some info about that repo.



4. If you want to make changes to the code and have your own copy of it to register, clone the code into a public repository in Github or in MAAP Gitlab.

If you would like to create a template from an existing repository, create a new repository on GitHub where you have the option to “Import a repository” where you can supply the old repository git link.



Then, clone that new repository into the ADE via the steps above

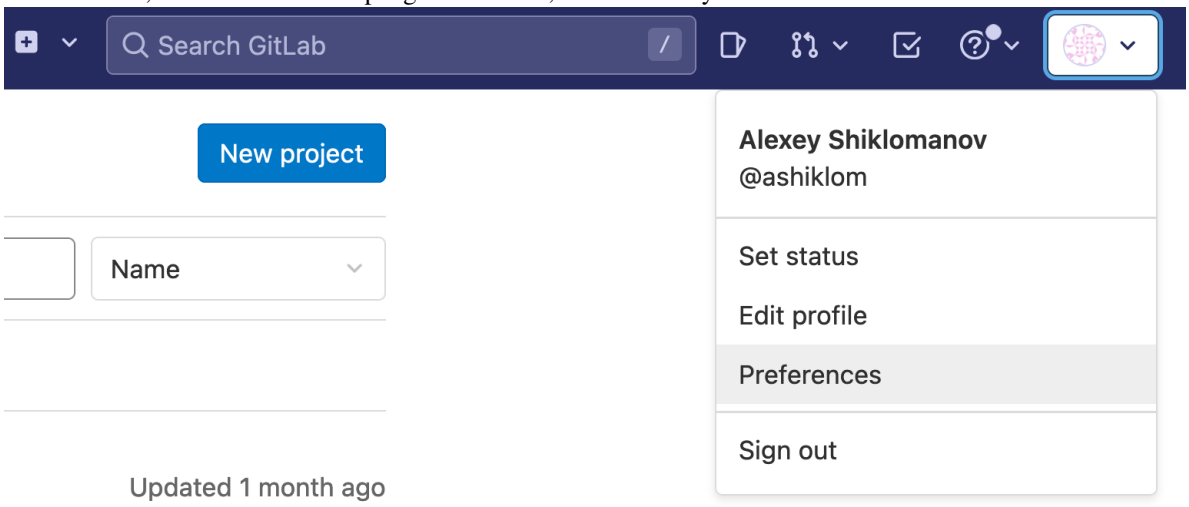
4.8.2 Working with the MAAP GitLab

Note: git can behave slowly and strangely over s3 bucket-based storage (i.e., `my-private-bucket` and `my-public-bucket`). It is recommended to set up your git-tracked repos on the root (somewhere inside of `~` or `/projects`).

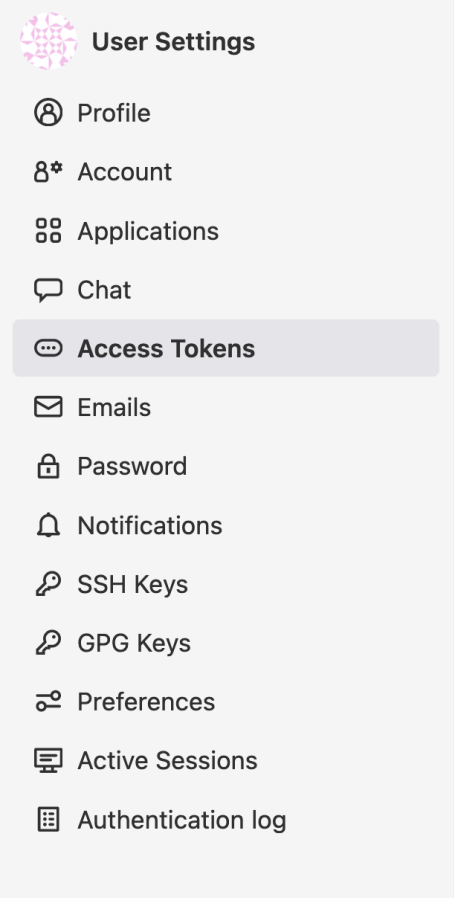
The MAAP GitLab instance is located at https://repo.maap-project.org/ . Make sure you can access this from the browser using your MAAP (EarthData Login) credentials.

For NASA security reasons, MAAP cannot communicate with its GitLab instance over SSH. There also isn't a username-password authentication option. Therefore, the recommended way to access MAAP repositories is to use GitLab Personal Access Tokens.

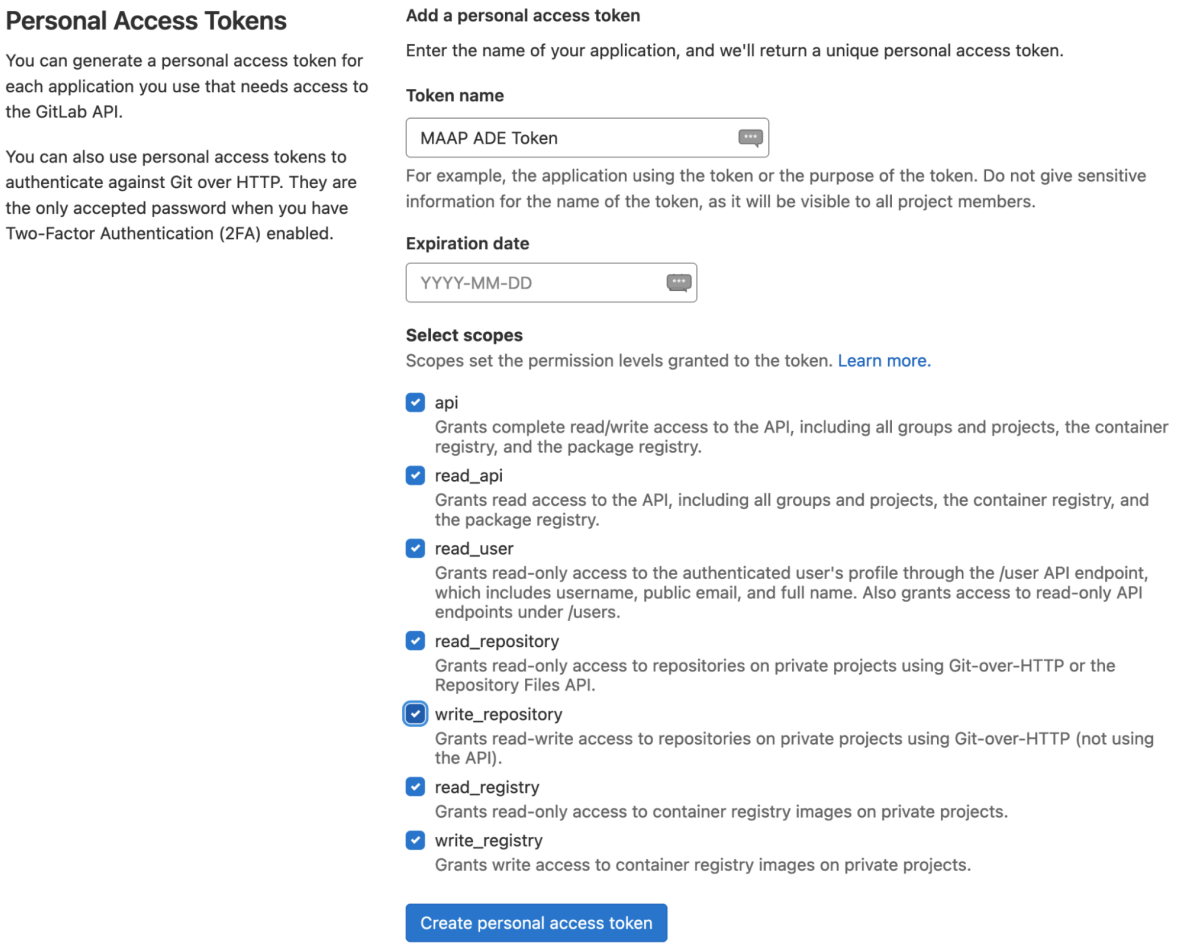
1. In GitLab, in the top-right corner, click your user icon → "Preferences"



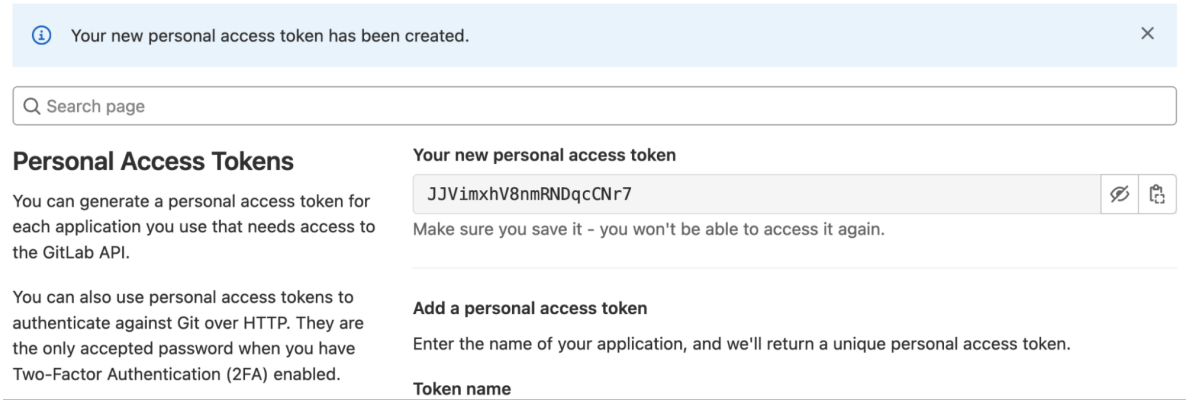
2. In the "User settings" menu, navigate to "Access Tokens".



3. Create a new token with at least “read_repository” and “write_repository” permissions.



4. After clicking "create personal access token", you'll see a message like this pop up. Make sure you copy this token into a text file — you will not be able to access it again.



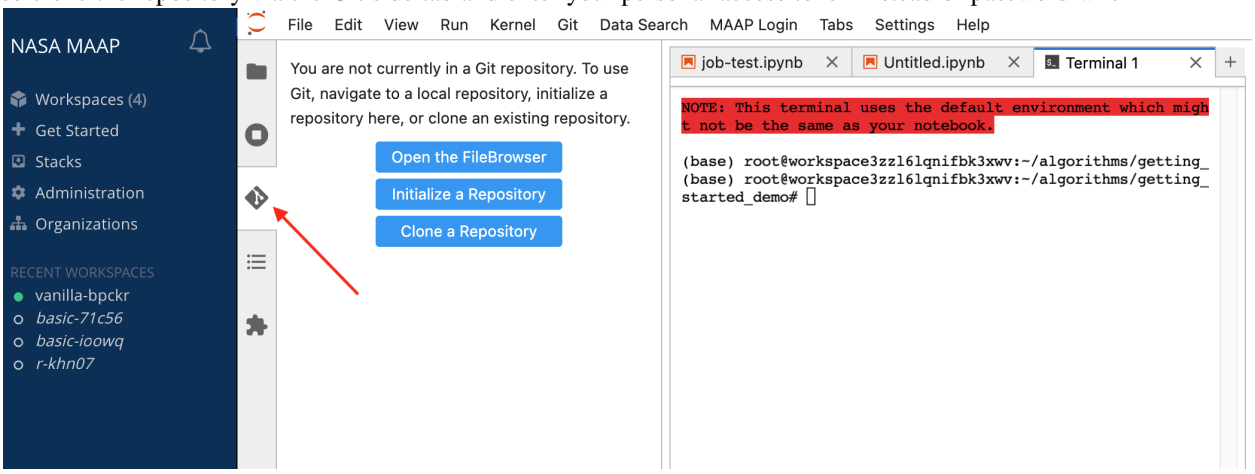
5. In the MAAP ADE, include this access token as part of the remote URL; e.g.,

```
git clone https://username:AccessToken@repo.maap-project.org/username/repo_name
```

For example:

```
git clone https://ashiklom:JJVimxhV8nmRNDqcCNr7@repo.maap-project.org/ashiklom/fireatlas
```

You can also clone the repository via the Git side tab and enter your personal access token instead of password when prompted.



If you want to use multiple code repositories, it's possible to configure a repository to have multiple remotes — e.g.,

To add the `maap` remote and set the URL, cd into the repo and use this:

```
git remote add maap https://username:AccessToken@repo.maap-project.org/username/repo_name
```

If you already have the remote called `maap` set up, you can set the remote URL using this instead:

```
git remote set-url maap https://username:AccessToken@repo.maap-project.org/username/repo_
↪name
```

Then, you can use these commands to push your code and effectively synchronize between Github and MAAP GitLab (for algorithm registration):

Push to Github:

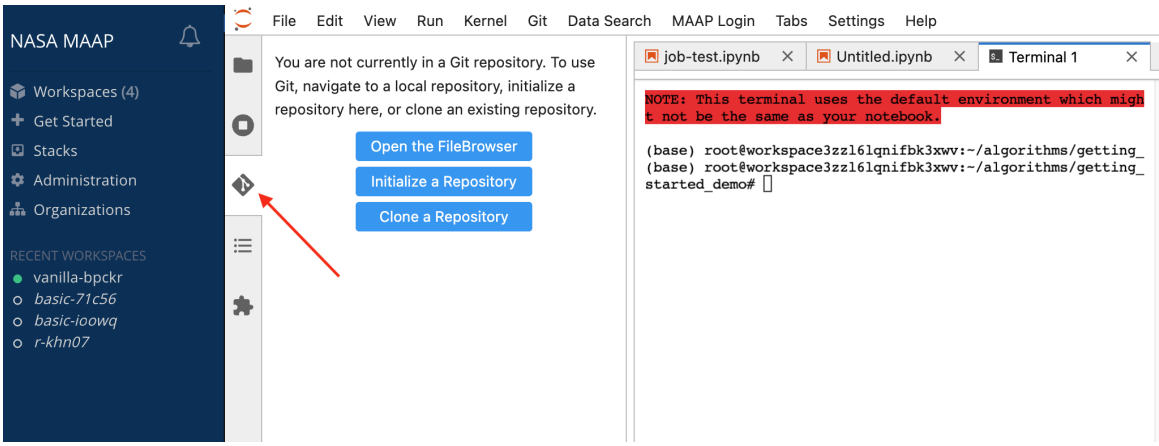
```
git push origin <branch name>
```

Push to MAAP GitLab:

```
git push maap <branch name>
```

4.8.3 Initializing a new repository

Note It is recommended to create a new repository on Git, then clone it and paste files in, especially if you are starting from scratch and don't have any files yet

1. Navigate to the folder you would like to make a Git repository
2. Click the Git sidebar 

3. Select "Initialize a Repository"
4. Create a new, empty repository on GitHub or MAAP GitLab instance with the same name as your folder
5. Add a remote with the following code in your terminal

```
git remote add origin https://github.com/username/repo_name.git
git remote set-url origin https://github.com/username/repo_name.git
```

Or if you are using MAAP GitLab

```
git remote add maap https://username:AccessToken@repo.maap-project.org/username/repo_name
git remote set-url maap https://username:AccessToken@repo.maap-project.org/username/repo_
↪name
```

And make code changes with

```
git add .
git commit -m <message>
git push --set-upstream origin <branch>
```

And enter classic personal access token instead of password when prompted

4.8.4 Committing Changes to a Git Repository

Before you update your project with your changes, you need to get a classic personal access token from GitHub. MAAP currently doesn't supported fine-grained tokens. Create a classic personal access token by following these steps in Developer settings. Remember to save your personal access token for later use.

Jupyter notebook checkpoints bloat git unnecessarily, so you can add the following to your `.gitignore` file to prevent this:

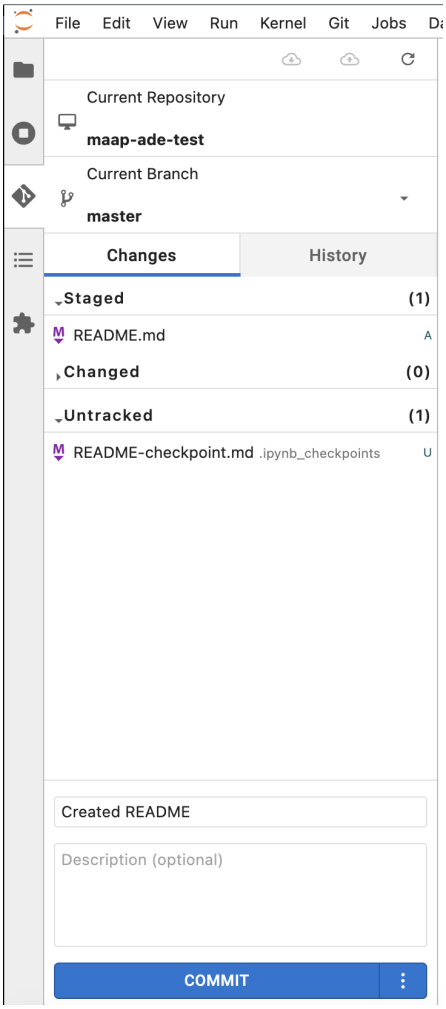
```
.ipynb_checkpoints
*/.ipynb_checkpoints/*
```

For more information on `.gitignore` files, see here.

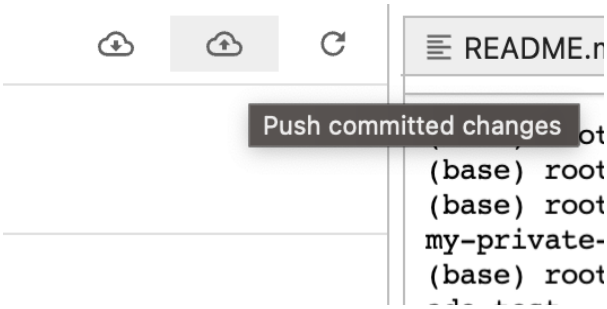
Using the left side panel in the Jupyter interface, you can push changes to your Git project whether it is on the MAAP GitLab, or a public repository on GitHub.

If you are more comfortable using the command line to interact with Git, you do not need to use the side panel. It will work the same way in the terminal, once you navigate to the project's filepath.

When you are ready to update your project with your changes, navigate to the Git panel. Add the files you want to change to the list of staged changes by clicking the + to the right of the file name. Then write a commit message, and blue commit button.



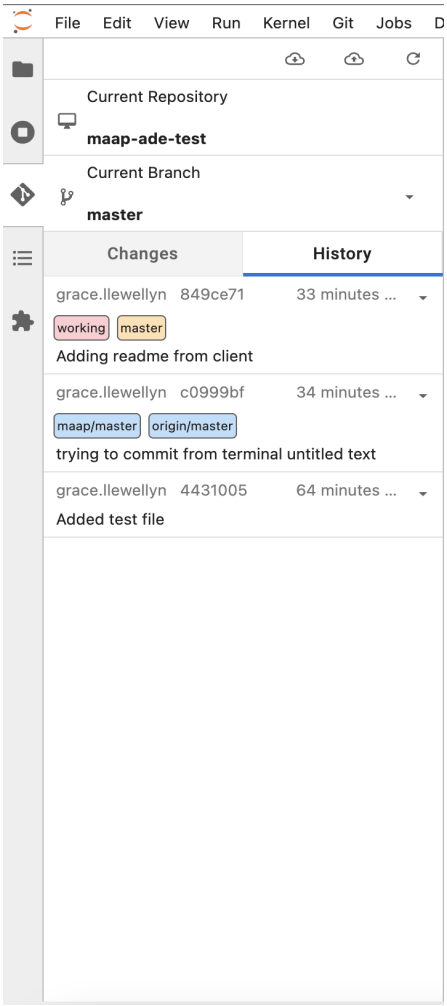
Now you need to push your changes by selecting the push changes button on the toolbar in the upper right.



You then may be prompted for your Git username and personal access token that you got above

You should then see “Successfully pushed”

If you want to check your commit history, look at branches, and confirm your updates have been pushed, you can see this on the history tab.



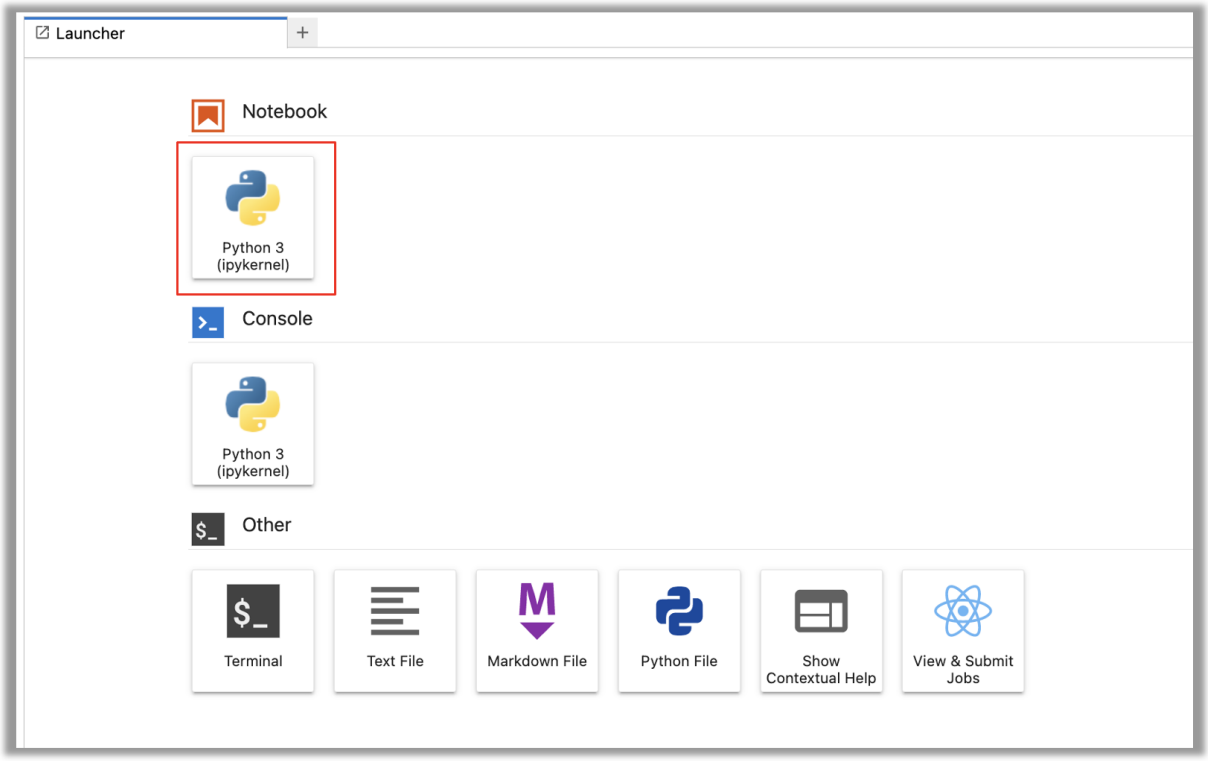
4.9 ADE Custom Extensions

4.9.1 MAAP Libraries

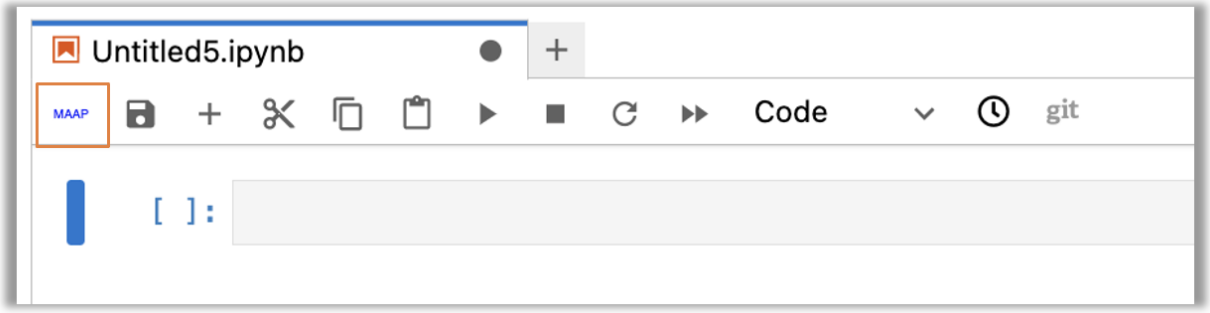
This Jupyter extension generates boilerplate code that imports and initializes MAAP libraries for usage in Jupyter notebooks. It currently supports the maap-py client library.

Access

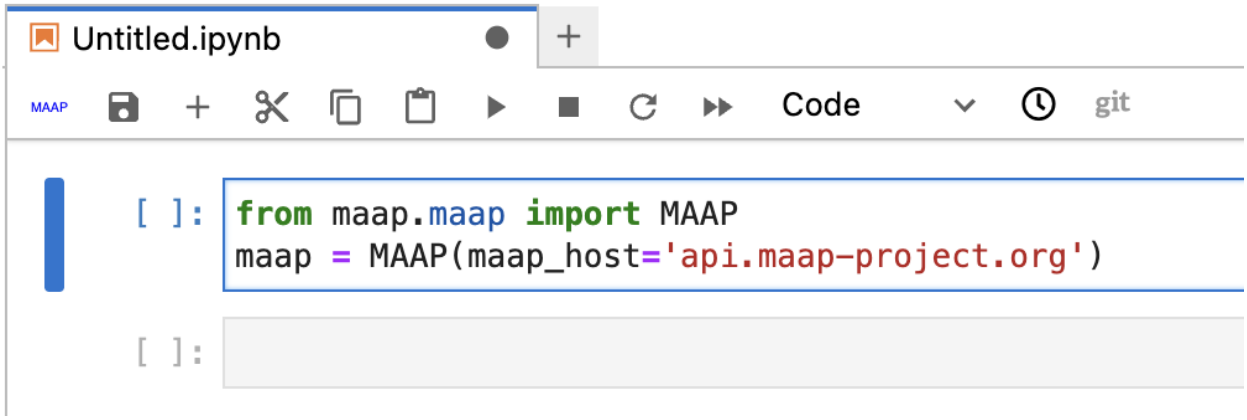
1. From your workspace, create a Jupyter notebook from the launcher pane.



2. Click on the **MAAP** icon.



3. The Jupyter notebook will be populated with the following code block. Run this code block to use functions from the maap-py client library.

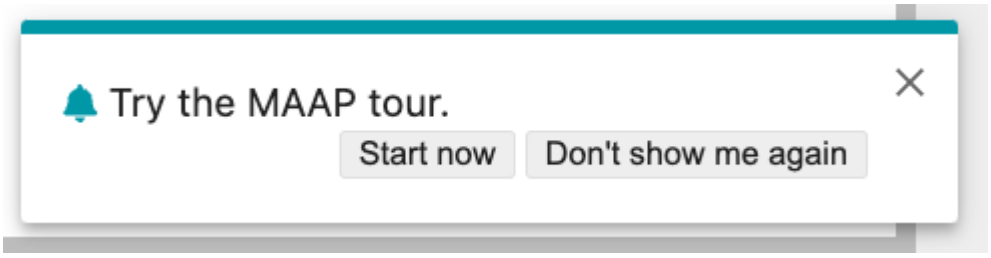


4.9.2 Maap Help Jupyter Extension

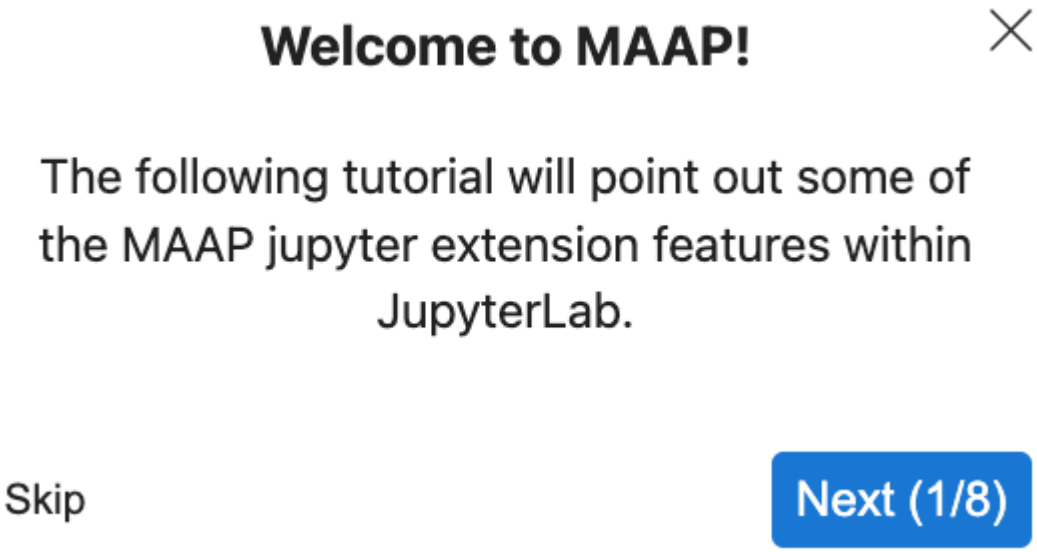
This Jupyter extension provides an interactive tutorial on a new user's first launch of the ADE and adds MAAP specific information to the help tab. The interactive tutorial can be accessed again from the help menu.

Interactive Tour

1. Launch the ADE for your first time as a new user, and see the MAAP tour notification



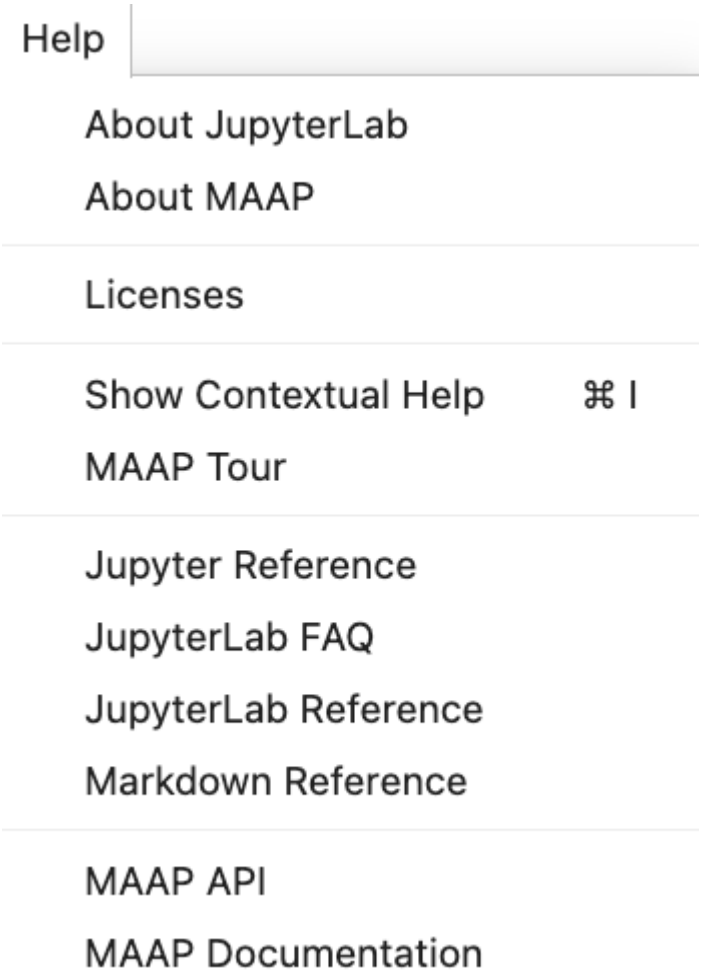
2. Step through the tour with the "Next" button.



3. The MAAP tour will no longer appear on launch if you a) finish the tour, b) press skip during the tour, or c) press "Don't show me again." You can revisit the MAAP tour by selecting it from the help menu.

MAAP Help Menu

Options show MAAP specific help information along with default JupyterLab help information.



4.10 FAQ

4.10.1 Which Files In My Workspace Are Persistent?

When you first start your workspace, there should be at least three folders already in the file browser: `my-private-bucket`, `my-public-bucket`, and `shared-buckets`.

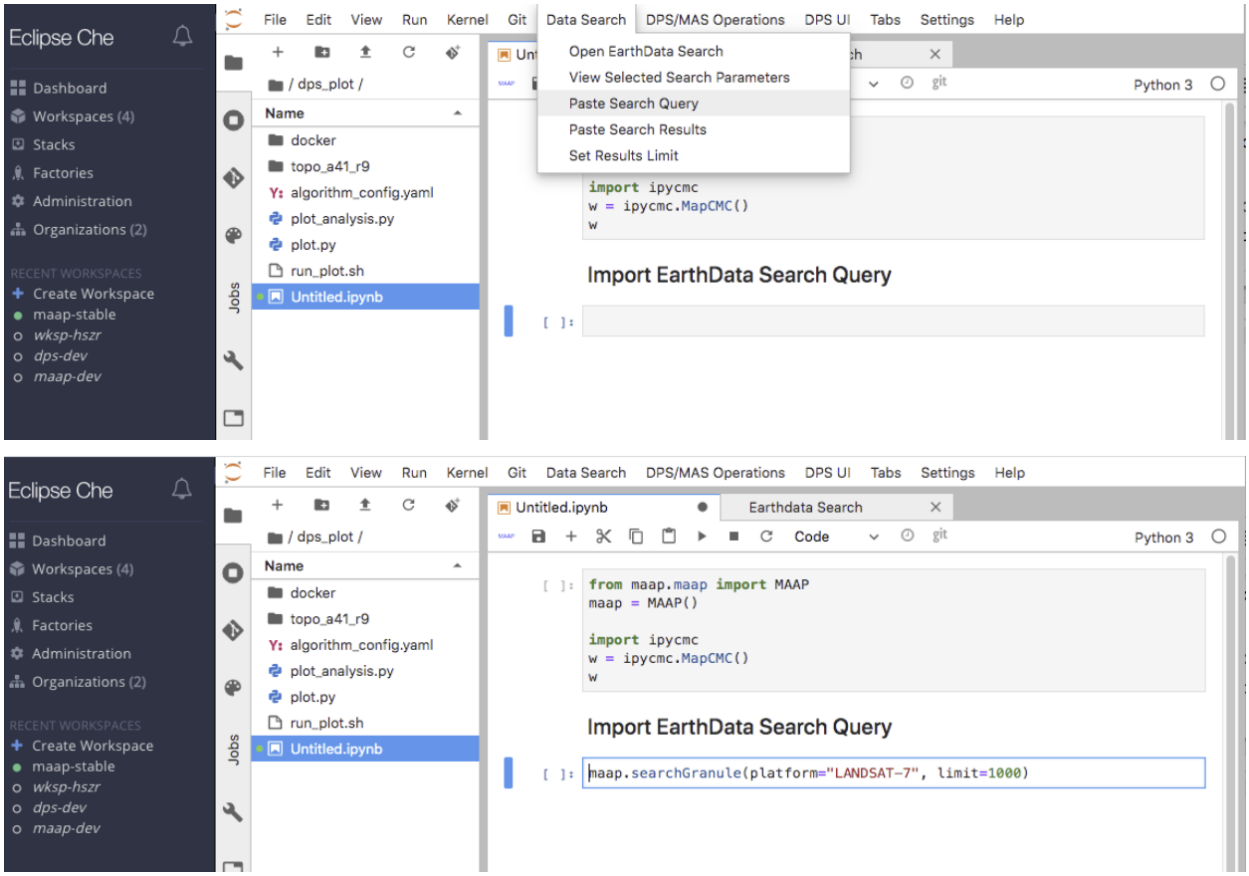
- `my-private-bucket` is an S3 bucket with persistent storage, but only visible to you. However, when you share your workspace with another user, both users' personal folders will be mounted and accessible from the shared workspace.
- `my-public-bucket` is an S3 bucket with persistent storage. It is the same as `~/shared-buckets/<my_username>/` — anything you put in here will be accessible to other users via `~/shared-buckets/`

<my_username> as a read-only file. Likewise, to find shared files from another user, look in ~/shared-buckets/<their_username>.

Anything placed within a s3-backed folder will be added to s3 and be persistent. Anything outside those folders will be ephemeral. Please clone your git repositories (e.g., dps_plot) outside the s3-backed folders, as they should already be version controlled elsewhere.

4.10.2 How Do I Copy My EARTHDATA Search Into My Jupyter Notebook?

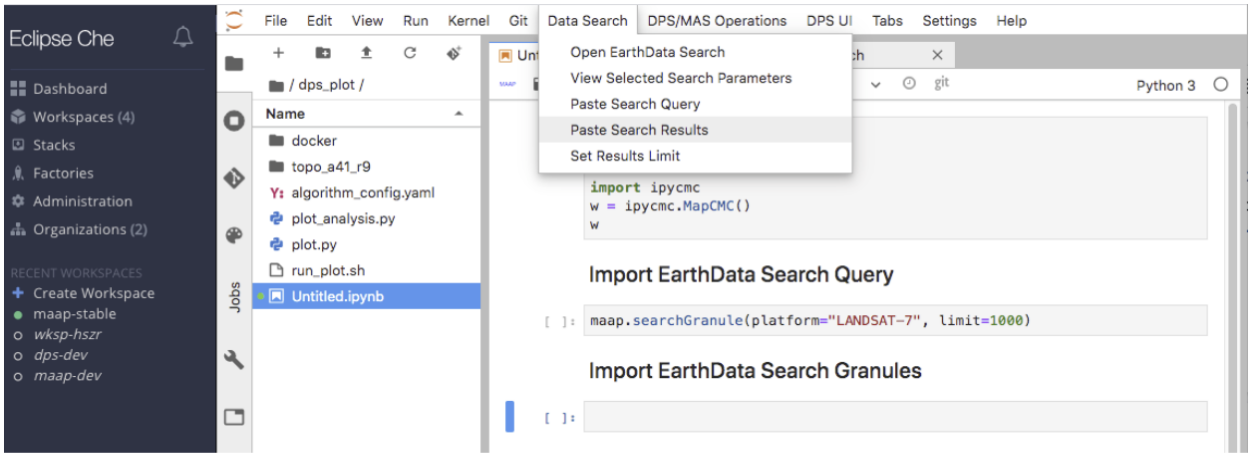
Launch EarthData Search by Data Search-> Open EarthData Search. After setting your search parameters or adding layers using the green plus, switch tabs back to your Jupyter notebook. At the top, open the Data Search menu, and select *Paste Search Query*.

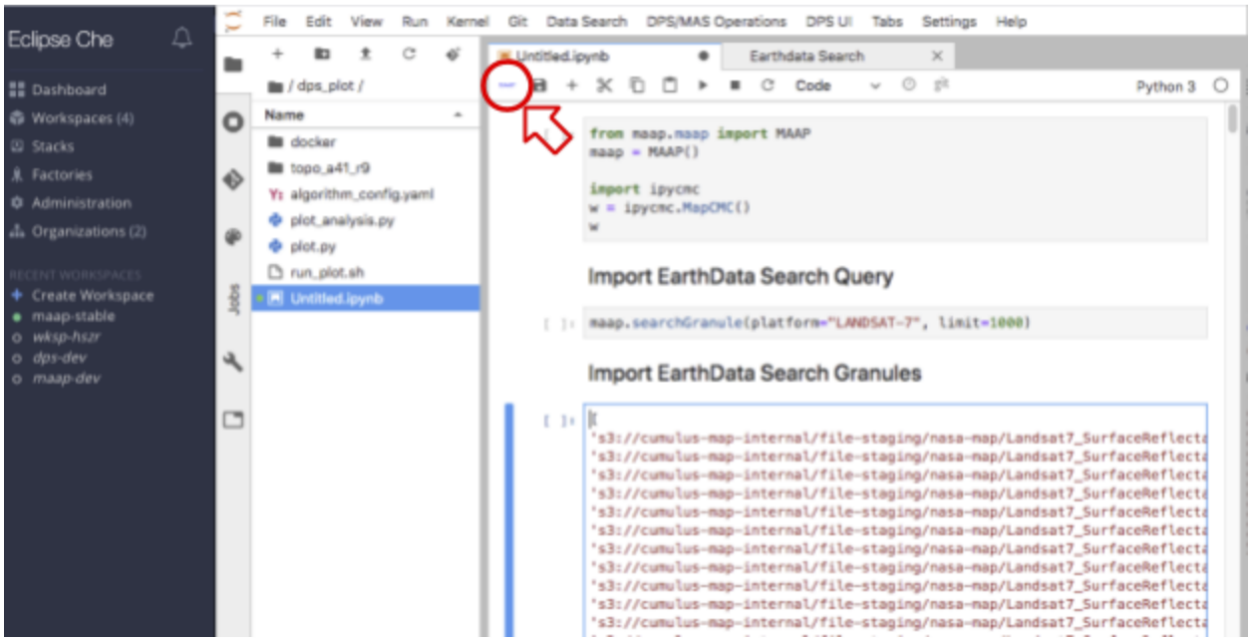


Caveat: This call uses the MAAP Python library. Make sure you import it before running the inserted code. You can do this by clicking on the blue "MAAP" text just below your notebook name.

4.10.3 How Do I Import Granules Over From My EARTHDATA Search Into My Jupyter Notebook?

Launch EarthData Search by Data Search-> Open EarthData Search. After setting your search parameters or adding layers using the green plus, switch tabs back to your Jupyter notebook. At the top, open the Data Search menu, and select *Paste Search Results*.





Caveat: This call uses the MAAP Python library. Make sure you import it before running the inserted code. You can do this by clicking on the blue "MAAP" text just below your notebook name (circled in red).

4.10.4 What is the Jupyter Server Extension?

This is the backend extension for the MAAP common collection of custom Jupyter extensions. This backend is accessible to users and other Jupyter extensions through a RESTful interface. The supported endpoints interact with the MAAP API and user workspace. For a list of the endpoints, refer to the repo documentation.

4.10.5 How do I request new data to be available in MAAP?

Authors: Emile Tenezakis (DevSeed), Sheyenne Kirkland (UAH)

Date: September 27, 2023

Description: in this example, we'll show you how to access MAAP's Data Request Form in Github, as well as provide an example on how to fill out and submit it. The purpose of this form is to provide a way for users to request data to be ingested into the MAAP platform for use.

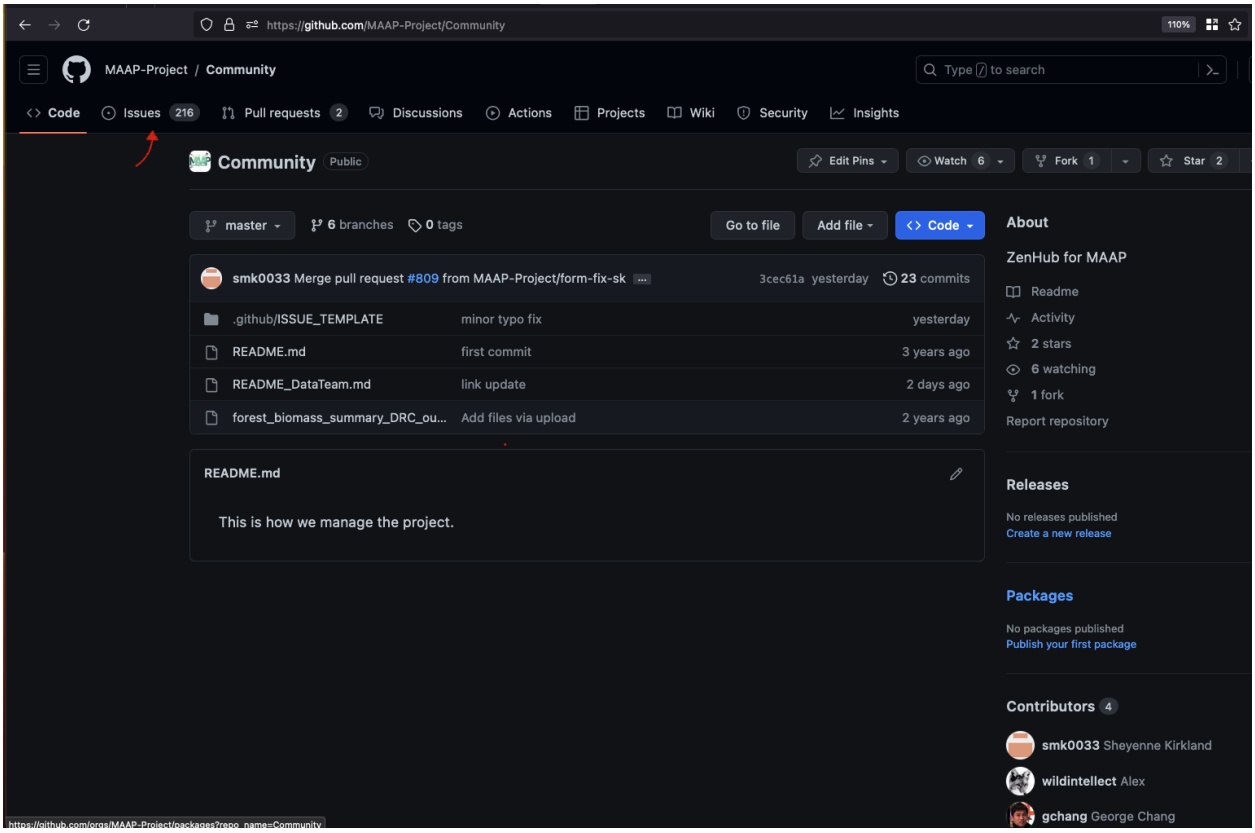
Accessing the Form

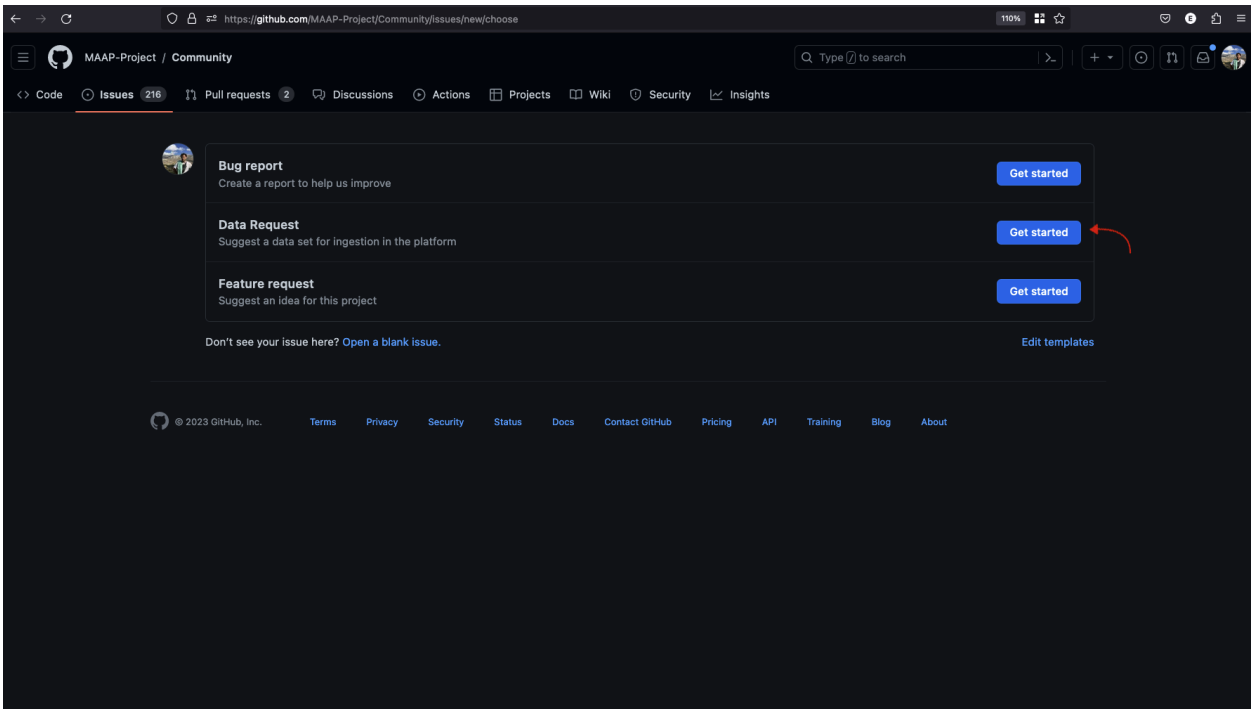
Direct Access Link

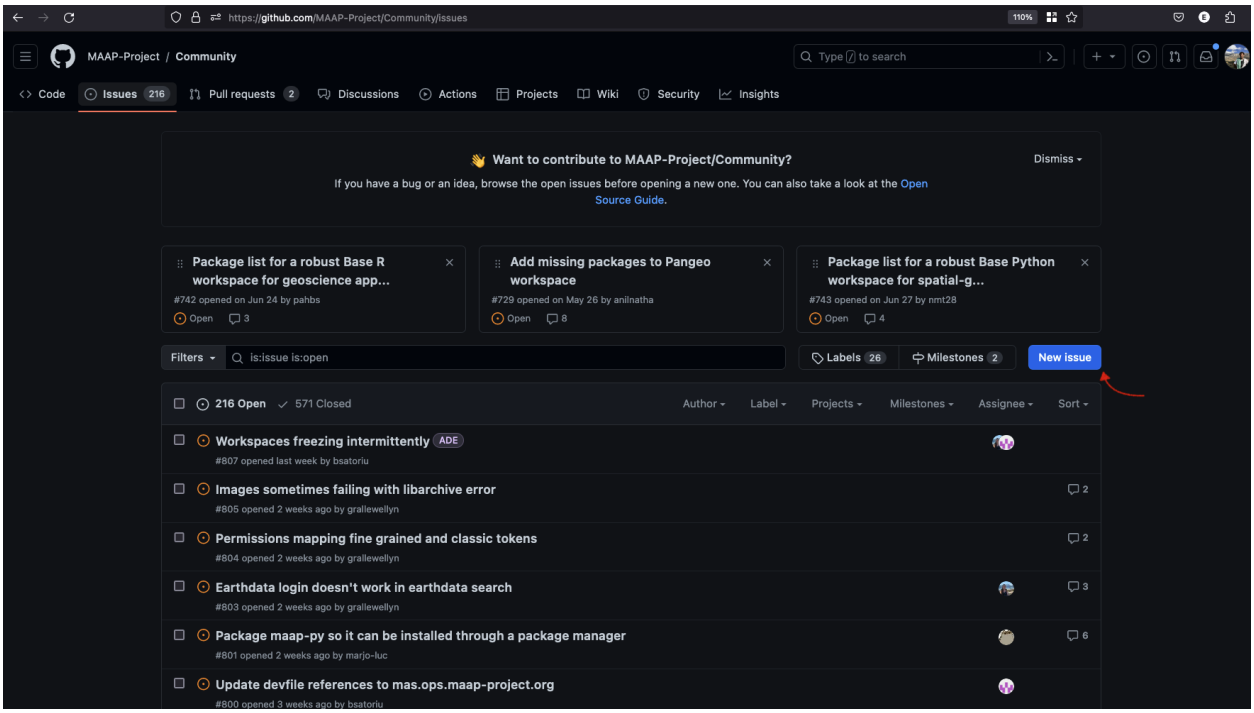
You can directly access the form through this link.

Navigate Manually to the Form

The form is hosted as an "issue template" in the MAAP Community Github repository. Navigate to this repository. From there, go to "Issues", and then click on "New Issue". There are three different options for users to open an issue. To request data, select "Get Started" next to "Data Request".

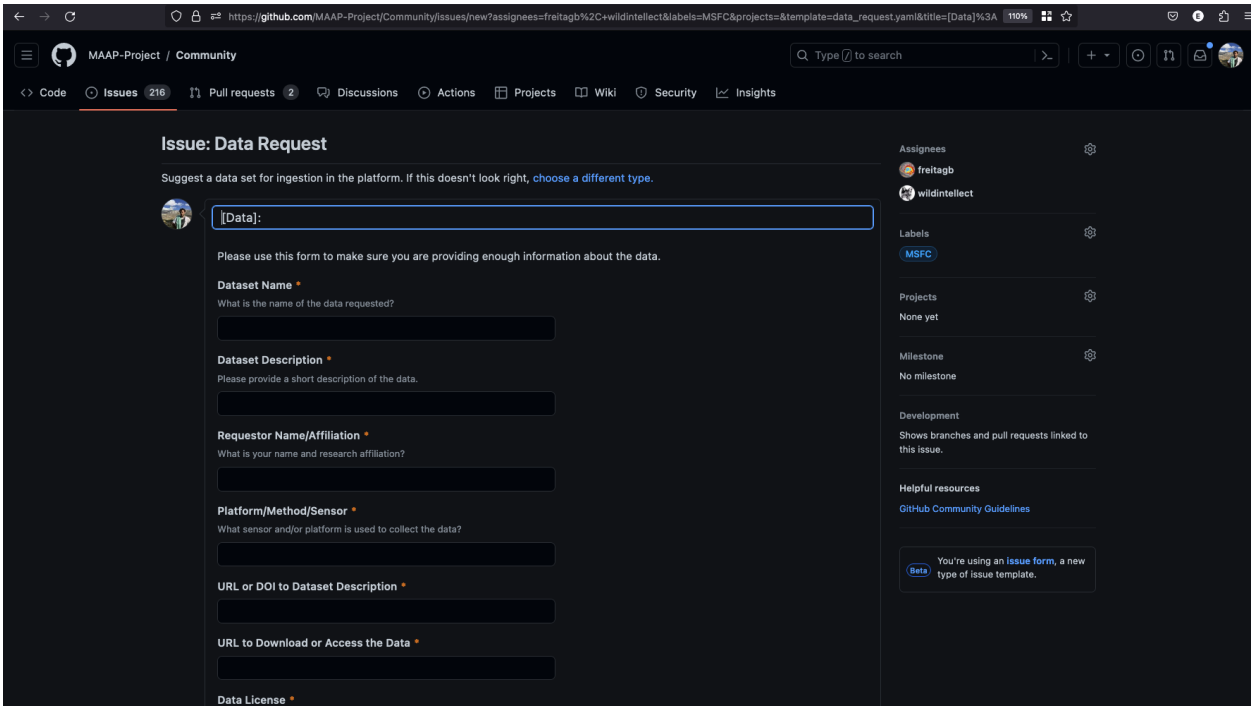

https://github.com/MAAP-Project/Community
MAAP-Project / Community
Type / to search
Code
Issues 216
Pull requests 2
Discussions
Actions
Projects
Wiki
Security
Insights
Community
Public
Edit Pins
Watch 6
Fork 1
Star 2
master
6 branches
0 tags
Go to file
Add file
Code
smk0033 Merge pull request #809 from MAAP-Project/form-fix-sk
3cec61a yesterday
23 commits
.github/ISSUE_TEMPLATE
minor typo fix
yesterday
README.md
first commit
3 years ago
README_DataTeam.md
link update
2 days ago
forest_biomass_summary_DRC_ou...
Add files via upload
2 years ago
README.md
This is how we manage the project.
About
ZenHub for MAAP
Readme
Activity
2 stars
6 watching
1 fork
Report repository
Releases
No releases published
Create a new release
Packages
No packages published
Publish your first package
Contributors 4
smk0033 Sheyenne Kirkland
wildintellect Alex
gchang George Chang


https://github.com/MAAP-Project/Community/issues/new/choose
MAAP-Project / Community
Type / to search
Code
Issues 216
Pull requests 2
Discussions
Actions
Projects
Wiki
Security
Insights
Bug report
Create a report to help us improve
Get started
Data Request
Suggest a data set for ingestion in the platform
Get started
Feature request
Suggest an idea for this project
Get started
Don't see your issue here? Open a blank issue.
Edit templates
© 2023 GitHub, Inc.
Terms
Privacy
Security
Status
Docs
Contact GitHub
Pricing
API
Training
Blog
About



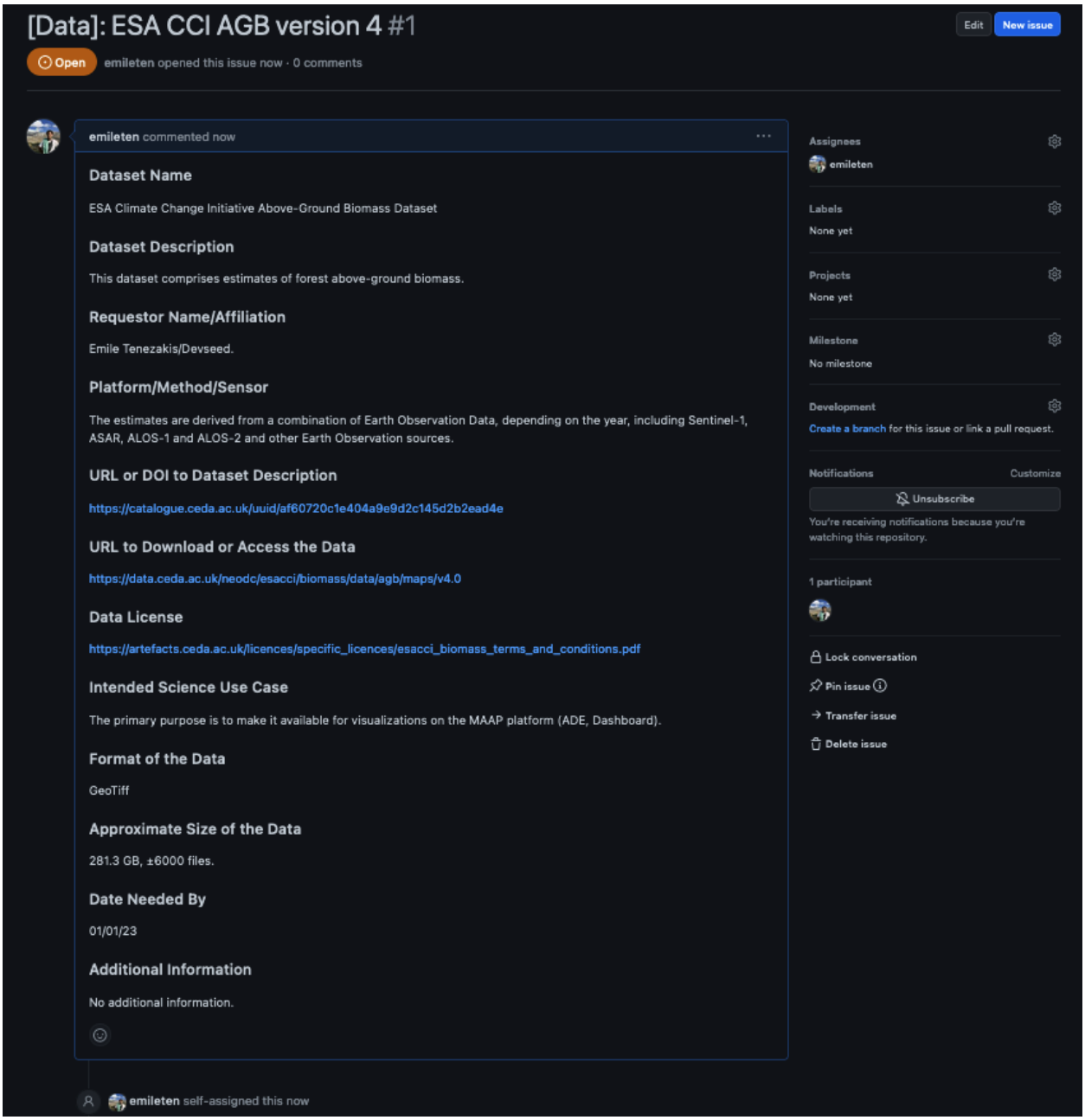
Filling out the Form

Once you open the form, you will see several fields to complete, some of which are required (marked with a red *) and some optional. This helps ensure that the data team recieves enough information. Assignees and labels are also automatically included, so all users have to do is enter information about the data.



Submission

Click on 'Submit new issue'. We've provided a screenshot of an example submitted issue below :



CHAPTER

FIVE

RELEASE NOTES

5.1 3.1.0

August 4, 2023
Hotfix to handle bugs that make working in the "new" ops ADE difficult

5.1.1 Added

- Added more capacity to the new ADE cluster to support more concurrent users.

5.1.2 Fixed

- Error with cursor jumping around in Jupyter & Opening blank notebook error (disable Jupyter collaboration feature): https://github.com/MAAP-Project/Community/issues/735
- Nested eclipse che menu error: https://github.com/MAAP-Project/Community/issues/733 (PR: https://github.com/MAAP-Project/maap-workspaces/pull/47)
- Add Show/Hide Che sidebar extension: https://github.com/MAAP-Project/Community/issues/692
- DPS notifications bug: https://github.com/MAAP-Project/Community/issues/778
- Maap libs extension can now show notifications: https://github.com/MAAP-Project/Community/issues/780
- Api_server now present in MAAP() instance (changing use of maapsec): https://github.com/MAAP-Project/Community/issues/781
- Open SSL fix: https://github.com/MAAP-Project/Community/issues/737
- Update Jupyter server to include API endpoints: https://github.com/MAAP-Project/Community/issues/685

5.1.3 Changed

5.1.4 Removed

- Remove ipycmc from default MAAP icon upper left of notebooks: https://github.com/MAAP-Project/Community/issues/779

CHAPTER

SIX

INDICES AND TABLES

- genindex
- modindex
- search