Sharing Work in Earth Engine: Basic UI and Apps (F6.3)

Author

Qiusheng Wu

Overview

The purpose of this chapter is to demonstrate how to design and publish Earth Engine Apps using both JavaScript and Python. You will be introduced to the Earth Engine User Interface JavaScript API and the *geemap* Python package. Upon completion of this chapter, you will be able to publish an Earth Engine App with a split-panel map for visualizing land cover change.

Learning Outcomes

- Designing a user interface for an Earth Engine App using JavaScript.
- Publishing an Earth Engine App for visualizing land cover change.
- Developing an Earth Engine App using Python and *geemap*.
- Deploying an Earth Engine App using a local computer as a web server.
- Publishing an Earth Engine App using Python and free cloud platforms.
- Creating a *conda* environment using Anaconda/Miniconda.
- Installing Python packages and using Jupyter Notebook.
- Commiting changes to a GitHub repository.

Assumes you know how to:

- Import images and image collections, filter, and visualize (Part F1).
- Use the basic functions and logic of Python.

Introduction to Theory

Earth Engine has a user interface API that allows users to build and publish interactive web apps directly from the JavaScript Code Editor. Many readers will have encountered a call to ui.Chart in other chapters, but much more interface functionality is available. In particular, users can utilize the `ui` functions to construct an entire graphical user interface (GUI) for their Earth Engine script. The GUI may include simple widgets (e.g., labels, buttons, checkboxes, sliders, text boxes) as well as more complex widgets (e.g., charts, maps, panels) for controlling the GUI layout. A complete list of the `ui` widgets and more information about panels can be found at the links below. Once a GUI is constructed, users can publish the App from the JavaScript Code Editor by clicking the **Apps** button above the script panel in the Code Editor.

- Widgets: https://developers.google.com/earth-engine/guides/ui_widgets
- Panels: https://developers.google.com/earth-engine/guides/ui_panels

Unlike the Earth Engine JavaScript API, the Earth Engine Python API does not provide functionality for building interactive user interfaces. Fortunately, the Jupyter ecosystem has *ipywidgets*, an architecture for creating interactive user interface controls (e.g., buttons, sliders, checkboxes, text boxes, dropdown lists) in Jupyter notebooks that communicate with Python code. The integration of graphical widgets into the Jupyter Notebook workflow allows users to configure ad hoc control panels to interactively sweep over parameters using graphical widget controls. One very powerful widget is the *output* widget, which can be used to display rich output generated by IPython, such as text, images, charts, and videos. A complete list of widgets and more information about the output widget can be found at the links below. By integrating *ipyleaflet* (for creating interactive maps) and *ipywidgets* (for designing interactive user interfaces), the *geemap* Python package (https://geemap.org) makes it much easier to explore and analyze massive Earth Engine datasets via a web browser in a Jupyter environment suitable for interactive exploration, teaching, and sharing. Users can build interactive Earth Engine Apps using *geemap* with minimal coding (Fig. F6.3.1).

- Widgets: https://ipywidgets.readthedocs.io/en/latest/examples/Widget%20List.html
- Output: https://ipywidgets.readthedocs.io/en/latest/examples/Output%20Widget.html

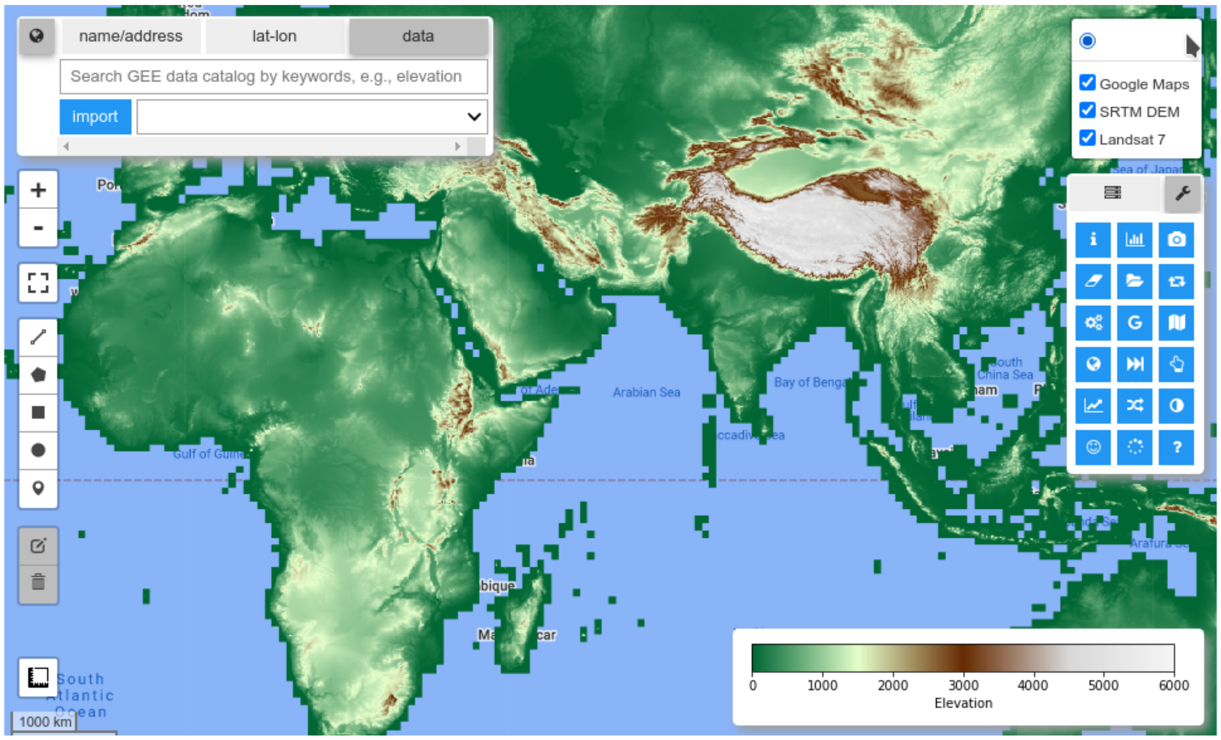


Fig. F6.3.1 The GUI of *geemap* in a Jupyter environment

Practicum

Section 1. Building an Earth Engine App Using JavaScript

If you have not already done so, you can add the book's code repository to the Code Editor by entering https://code.earthengine.google.com/?accept_repo=projects/gee-edu/book (or the short URL bit.ly/EEFA-repo) into your browser. The book's scripts will then be available in the script manager panel to view, run, or modify. If you have trouble finding the repo, you can visit bit.ly/EEFA-repo-help for help.

In this section, you will learn how to design a user interface for an Earth Engine App using JavaScript and the Earth Engine User Interface API. Upon completion of this section, you will have an Earth Engine App with a split-panel map for visualizing land cover change using the Landsat-based United States Geological Survey National Land Cover Database (NLCD).

First, let's define a function for filtering the NLCD `ImageCollection` by year and select the `landcover` band. The function returns an Earth Engine `ui.Map.Layer` of the `landcover` band of the selected NLCD image. Note that as of this writing, NLCD spans nine epochs: 1992, 2001, 2004, 2006, 2008, 2011, 2013, 2016, and 2019. The 1992 data are primarily based on unsupervised classification of Landsat data, while the rest of the images rely on the imperviousness data layer for the urban classes and on a decision-tree classification for the rest. The 1992 image is not directly comparable to any later editions of NLCD (see the Earth Engine Data Catalog for more details, if needed). Therefore, we will use only the eight epochs after 2000 in this lab.

```
// Get an NLCD image by year.
var getNLCD = function(year) {
    // Import the NLCD collection.
    var dataset = ee.ImageCollection(
        'USGS/NLCD_RELEASES/2019_REL/NLCD');

    // Filter the collection by year.
    var nlcd = dataset.filter(ee.Filter.eq('system:index', year))
        .first();

    // Select the land cover band.
    var landcover = nlcd.select('landcover');
    return ui.Map.Layer(landcover, {}, year);
};
```

Our intention is to create a dropdown list so that when a particular epoch is selected, the corresponding NLCD image layer will be displayed on the map. We'll define a dictionary with each NLCD epoch as the key and its corresponding NLCD image layer as the value. The keys of the dictionary (i.e., the eight NLCD epochs) will be used as the input to the dropdown lists (`ui.Select`) on the split-level map.

```
// Create a dictionary with each year as the key
// and its corresponding NLCD image layer as the value.
var images = {
    '2001': getNLCD('2001'),
    '2004': getNLCD('2004'),
    '2006': getNLCD('2006'),
    '2008': getNLCD('2008'),
    '2011': getNLCD('2011'),
    '2013': getNLCD('2013'),
    '2016': getNLCD('2016'),
    '2019': getNLCD('2019'),
};
```

The split-panel map is composed of two individual maps, `leftMap` and `rightMap`. The map controls (e.g., `zoomControl`, `scaleControl`, `mapTypeControl`) will be shown only on `rightMap`. A control panel (`ui.Panel`) composed of a label (`ui.Label`) and a dropdown list (`ui.Select`) is added to each map. When an NLCD epoch is selected from a dropdown list, the function `updateMap` will be called to show the corresponding image layer of the selected epoch.

```
// Create the left map, and have it display the first layer.
var leftMap = ui.Map();
leftMap.setControlVisibility(false);
var leftSelector = addLayerSelector(leftMap, 0, 'top-left');

// Create the right map, and have it display the last layer.
var rightMap = ui.Map();
rightMap.setControlVisibility(true);
var rightSelector = addLayerSelector(rightMap, 7, 'top-right');

// Adds a layer selection widget to the given map, to allow users to
// change which image is displayed in the associated map.
function addLayerSelector(mapToChange, defaultValue, position) {
    var label = ui.Label('Select a year:');

    // This function changes the given map to show the selected image.
    function updateMap(selection) {
        mapToChange.layers().set(0, images[selection]);
    }

    // Configure a selection dropdown to allow the user to choose
    // between images, and set the map to update when a user
    // makes a selection.
    var select = ui.Select({
        items: Object.keys(images),
        onChange: updateMap
    });
    select.setValue(Object.keys(images)[defaultValue], true);

    var controlPanel =
        ui.Panel({
            widgets: [label, select],
            style: {
                position: position
            }
        });

    mapToChange.add(controlPanel);
}
```

When displaying a land cover classification image on the **Map**, it would be useful to add a legend to make it easier for users to interpret the land cover type associated with each color. Let's define a dictionary that will be used to construct the legend. The dictionary contains two keys: `names` (a list of land cover types) and `colors` (a list of colors associated with each land cover type). The legend will be placed in the bottom right of the **Map**.

```
// Set the legend title.
var title = 'NLCD Land Cover Classification';

// Set the legend position.
var position = 'bottom-right';

// Define a dictionary that will be used to make a legend
var dict = {
    'names': [
        '11Open Water',
        '12Perennial Ice/Snow',
        '21Developed, Open Space',
        '22Developed, Low Intensity',
        '23Developed, Medium Intensity',
        '24Developed, High Intensity',
        '31Barren Land (Rock/Sand/Clay)',
        '41Deciduous Forest',
        '42Evergreen Forest',
        '43Mixed Forest',
        '51Dwarf Scrub',
        '52Shrub/Scrub',
        '71Grassland/Herbaceous',
        '72Sedge/Herbaceous',
        '73Lichens',
        '74Moss',
        '81Pasture/Hay',
        '82Cultivated Crops',
        '90Woody Wetlands',
        '95Emergent Herbaceous Wetlands',
    ],

    'colors': [
        '#466b9f', '#d1def8', '#dec5c5', '#d99282', '#eb0000',
        '#ab0000',
        '#b3ac9f', '#68ab5f', '#1c5f2c', '#b5c58f', '#af963c',
        '#ccb879',
        '#dfdfc2', '#d1d182', '#a3cc51', '#82ba9e', '#dcd939',
        '#ab6c28',
        '#b8d9eb', '#6c9fb8',
    ]
};
```

With the legend dictionary defined above, we can now create a panel to hold the legend widget and add it to the **Map**. Each row on the legend widget is composed of a color box followed by its corresponding land cover type.

```
// Create a panel to hold the legend widget.
var legend = ui.Panel({
    style: {
        position: position,
        padding: '8px 15px'
    }
});

// Function to generate the legend.
function addCategoricalLegend(panel, dict, title) {

    // Create and add the legend title.
    var legendTitle = ui.Label({
        value: title,
        style: {
            fontWeight: 'bold',
            fontSize: '18px',
            margin: '0 0 4px 0',
            padding: '0'
        }
    });
    panel.add(legendTitle);

    var loading = ui.Label('Loading legend...', {
        margin: '2px 0 4px 0'
    });
    panel.add(loading);

    // Creates and styles 1 row of the legend.
    var makeRow = function(color, name) {
        // Create the label that is actually the colored box.
        var colorBox = ui.Label({
            style: {
                backgroundColor: color,
                // Use padding to give the box height and width.
                padding: '8px',
                margin: '0 0 4px 0'
            }
        });

        // Create the label filled with the description text.
        var description = ui.Label({
            value: name,
            style: {
                margin: '0 0 4px 6px'
            }
        });

        return ui.Panel({
            widgets: [colorBox, description],
            layout: ui.Panel.Layout.Flow('horizontal')
        });
    };

    // Get the list of palette colors and class names from the image.
    var palette = dict.colors;
    var names = dict.names;
    loading.style().set('shown', false);

    for (var i = 0; i < names.length; i++) {
        panel.add(makeRow(palette[i], names[i]));
    }

    rightMap.add(panel);

}
```

The last step is to create a split-panel map to hold the linked maps (`leftMap` and `rightMap`) and tie everything together. When users pan and zoom one map, the other map will also be panned and zoomed to the same extent automatically. When users select a year from a dropdown list, the image layer will be updated accordingly. Users can use the slider to swipe through and visualize land cover change easily (Fig. F6.3.2). Please make sure you minimize the Code Editor and maximize the **Map** so that you can see the dropdown widget in the upper-right corner of the map.

```
addCategoricalLegend(legend, dict, title);

// Create a SplitPanel to hold the adjacent, linked maps.
var splitPanel = ui.SplitPanel({
    firstPanel: leftMap,
    secondPanel: rightMap,
    wipe: true,
    style: {
        stretch: 'both'
    }
});

// Set the SplitPanel as the only thing in the UI root.
ui.root.widgets().reset([splitPanel]);
var linker = ui.Map.Linker([leftMap, rightMap]);
leftMap.setCenter(-100, 40, 4);
```

Code Checkpoint F63a. The book’s repository contains a script that shows what your code should look like at this point.

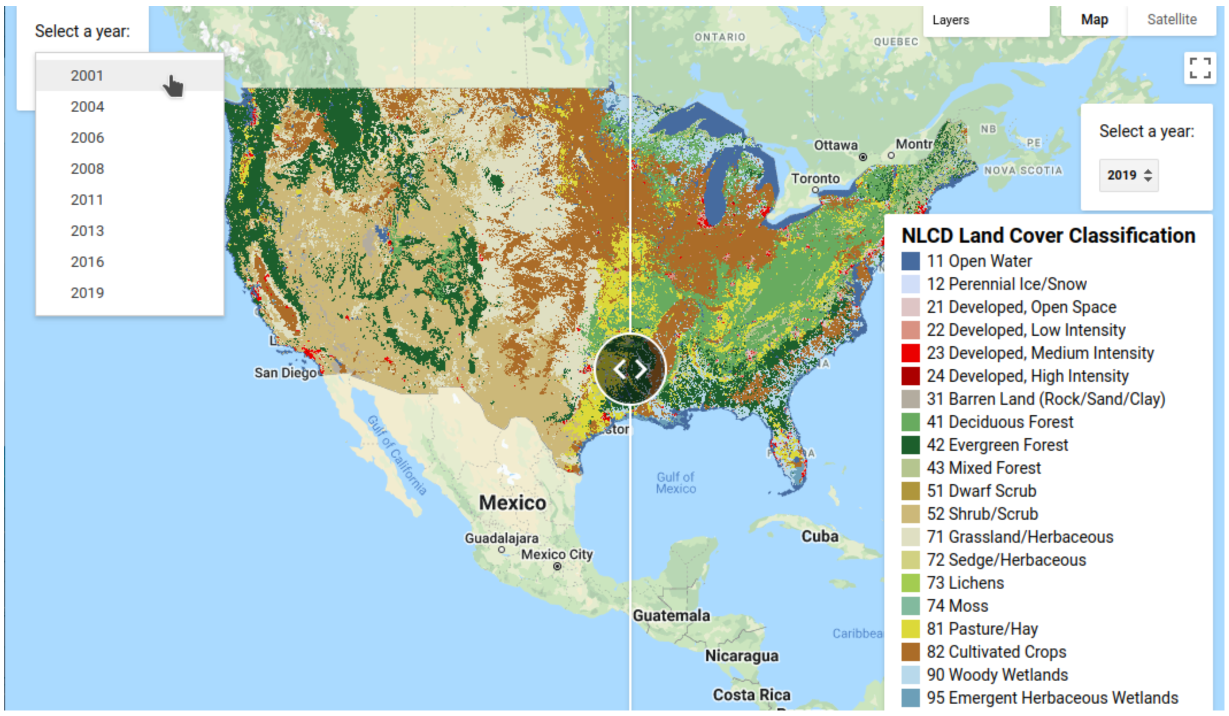


Fig. F6.3.2 A split-panel map for visualizing land cover change using NLCD

<u>Section 2. Publishing an Earth Engine App from the Code Editor</u>

The goal of this section is to publish the Earth Engine App that we created in Sect. 1. The look and feel of interfaces changes often; if the exact windows described below change over time, the concepts should remain stable to help you to publish your App. First, load the script (see the Code Checkpoint in Sect. 1) into the Code Editor. Then, open the **Manage Apps** panel by clicking the **Apps** button above the script panel in the Code Editor (Fig. F6.3.3).

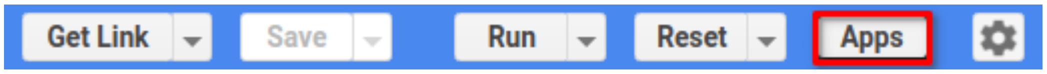


Fig. F6.3.3 The **Apps** button in the JavaScript Code Editor

Now click on the **New App** button (Fig. F6.3.4).

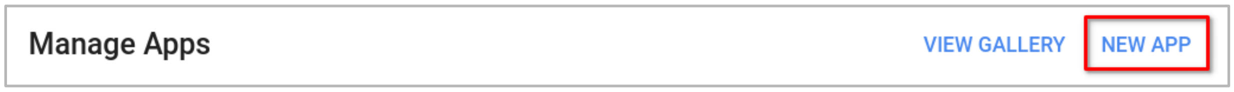
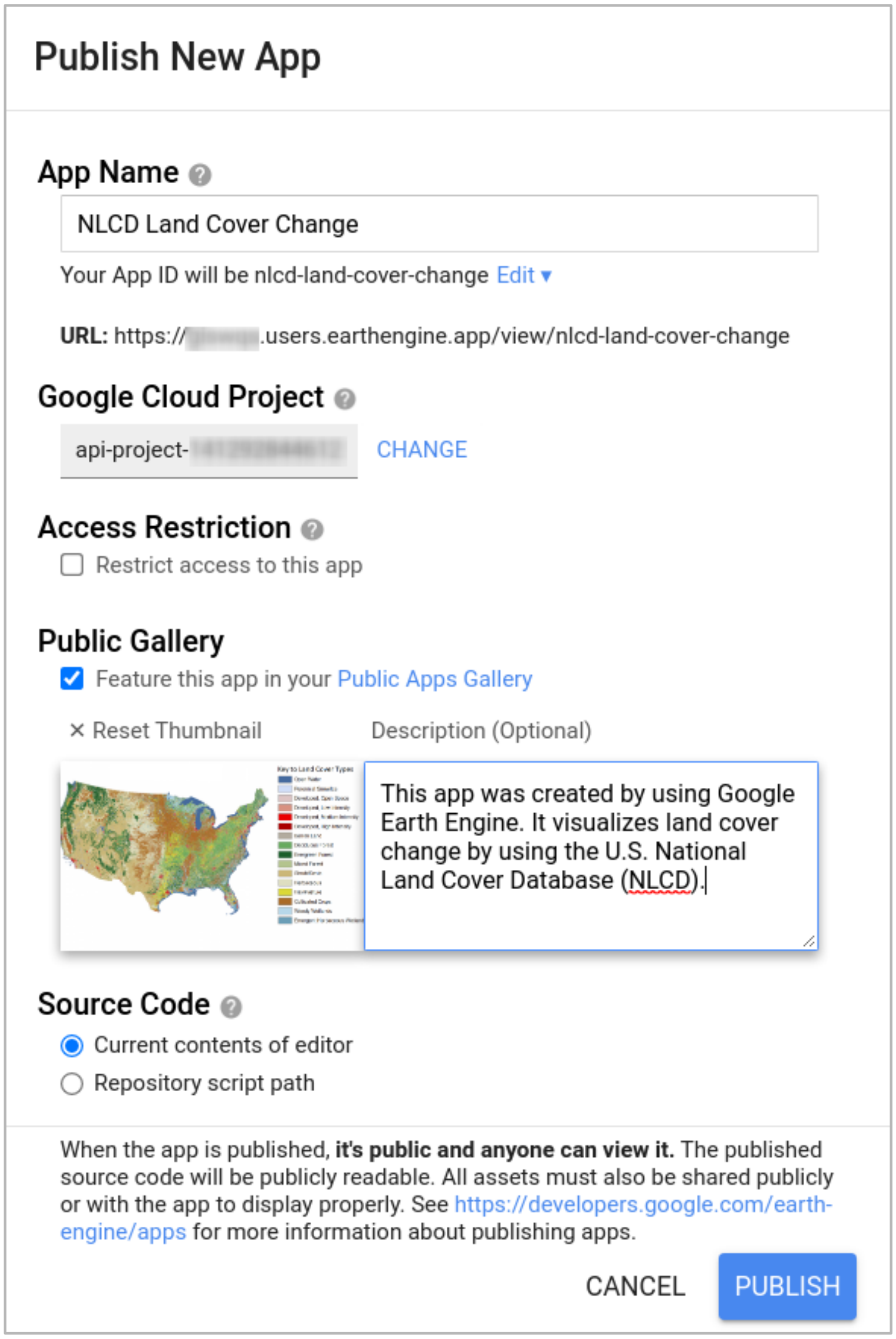


Fig. F6.3.4 The **New App** button

In the **Publish New App** dialog (Fig. F6.3.5), choose a name for the App (e.g., NLCD Land Cover Change), select a Google Cloud Project, provide a thumbnail to be shown in the Public Apps Gallery, and specify the location of the App's source code. You may restrict access to the App to a particular Google Group or make it publicly accessible. Check **Feature this app in your Public Apps Gallery** if you would like this App to appear in your public gallery of Apps available at https://USERNAME.users.earthengine.app. When all fields are filled out and validated, the **Publish** button will be enabled; click it to complete publishing the App.



Publish New App

App Name

NLCD Land Cover Change

Your App ID will be nlcd-land-cover-change Edit ▾

URL: https://[redacted].users.earthengine.app/view/nlcd-land-cover-change

Google Cloud Project

api-project-[redacted] CHANGE

Access Restriction

☐ Restrict access to this app

Public Gallery

☑ Feature this app in your Public Apps Gallery

× Reset Thumbnail

Description (Optional)

This app was created by using Google Earth Engine. It visualizes land cover change by using the U.S. National Land Cover Database (NLCD).

Source Code

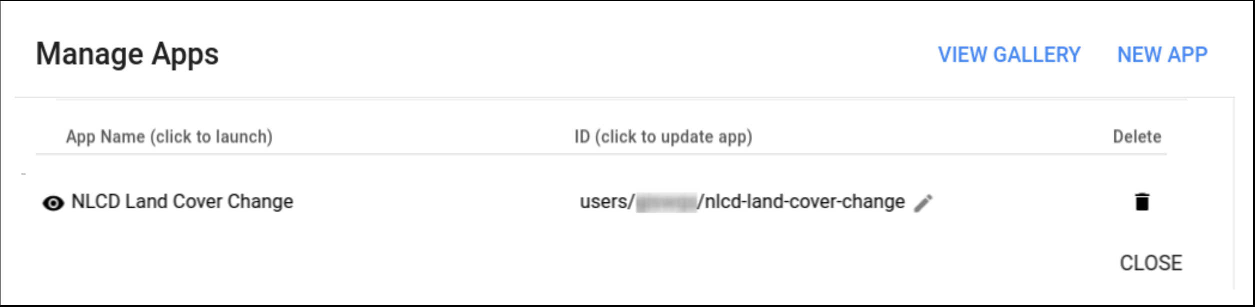
[✓ Current contents of editor]

Repository script path

When the app is published, **it's public and anyone can view it.** The published source code will be publicly readable. All assets must also be shared publicly or with the app to display properly. See https://developers.google.com/earth-engine/apps for more information about publishing apps.

Fig. F6.3.5 The **Publish New App** dialog

To manage an App from the Code Editor, open the **Manage Apps** panel (Fig. F6.3.6) by clicking the **Apps** button above the script panel in the Code Editor (Fig. F6.3.3). There, you can update your App's configuration or delete the App.


Manage Apps VIEW GALLERY NEW APP

App Name (click to launch)	ID (click to update app)	Delete
NLCD Land Cover Change	users/[redacted]/nlcd-land-cover-change	

CLOSE

Fig. F6.3.6 The **Manage Apps** panel

Section 3. Developing an Earth Engine App Using geemap

In this section, you will learn how to develop an Earth Engine App using the *geemap* Python package and Jupyter Notebook. The *geemap* package is available on both PyPI (pip) and conda-forge. It is highly recommended that you create a fresh conda environment to install *geemap*.

Code Checkpoint F63b. The book's repository contains information about setting up a conda environment and installing *geemap*.

Once you have launched a Jupyter notebook in your browser, you can continue with the next steps of the lab.

On the Jupyter Notebook interface, click the **New** button in the upper-right corner and select **Python 3** to create a new notebook. Change the name of the notebook from "Untitled" to something meaningful (e.g., "nlcd_app"). With the newly created notebook, we can start writing and executing Python code.

First, let's import the Earth Engine and *geemap* libraries. Press Alt + Enter to execute the code and create a new cell below.

```
import ee
import geemap
```

Create an interactive map by specifying the map center (latitude, longitude) and zoom level (1–18). If this is the first time you use *geemap* and the Earth Engine Python API, a new tab will open in your browser asking you to authenticate Earth Engine. Follow the on-screen instructions to authenticate Earth Engine.

```
Map = geemap.Map(center=[40, -100], zoom=4)
Map
```

Retrieve the NLCD 2019 image by filtering the NLCD `ImageCollection` and selecting the `landcover` band. Display the NLCD 2019 image on the interactive **Map** by using `Map.addLayer`.

```
# Import the NLCD collection.
dataset = ee.ImageCollection('USGS/NLCD_RELEASES/2019_REL/NLCD')

# Filter the collection to the 2019 product.
nlcd2019 = dataset.filter(ee.Filter.eq('system:index',
'2019')).first()

# Select the land cover band.
landcover = nlcd2019.select('landcover')

# Display land cover on the map.
Map.addLayer(landcover, {}, 'NLCD 2019')
Map
```

Next, add the NLCD legend to the **Map**. The *geemap* package has several built-in legends, including the NLCD legend. Therefore, you can add the NLCD legend to the **Map** by using just one line of code (`Map.add_legend`).

```
title = 'NLCD Land Cover Classification'
Map.add_legend(legend_title=title, builtin_legend='NLCD')
```

Alternatively, if you want to add a custom legend, you can define a legend dictionary with labels as keys and colors as values, then you can use `Map.add_legend` to add the custom legend to the **Map**.

```
legend_dict = {
    '11 Open Water': '466b9f',
    '12 Perennial Ice/Snow': 'd1def8',
    '21 Developed, Open Space': 'dec5c5',
    '22 Developed, Low Intensity': 'd99282',
    '23 Developed, Medium Intensity': 'eb0000',
    '24 Developed High Intensity': 'ab0000',
    '31 Barren Land (Rock/Sand/Clay)': 'b3ac9f',
    '41 Deciduous Forest': '68ab5f',
    '42 Evergreen Forest': '1c5f2c',
    '43 Mixed Forest': 'b5c58f',
    '51 Dwarf Scrub': 'af963c',
    '52 Shrub/Scrub': 'ccb879',
    '71 Grassland/Herbaceous': 'dfdfc2',
    '72 Sedge/Herbaceous': 'd1d182',
    '73 Lichens': 'a3cc51',
    '74 Moss': '82ba9e',
    '81 Pasture/Hay': 'dcd939',
    '82 Cultivated Crops': 'ab6c28',
    '90 Woody Wetlands': 'b8d9eb',
    '95 Emergent Herbaceous Wetlands': '6c9fb8'
}
title = 'NLCD Land Cover Classification'
Map.add_legend(legend_title=title, legend_dict=legend_dict)
```

The **Map** with the NLCD 2019 image and legend should look like Fig. F6.3.7.

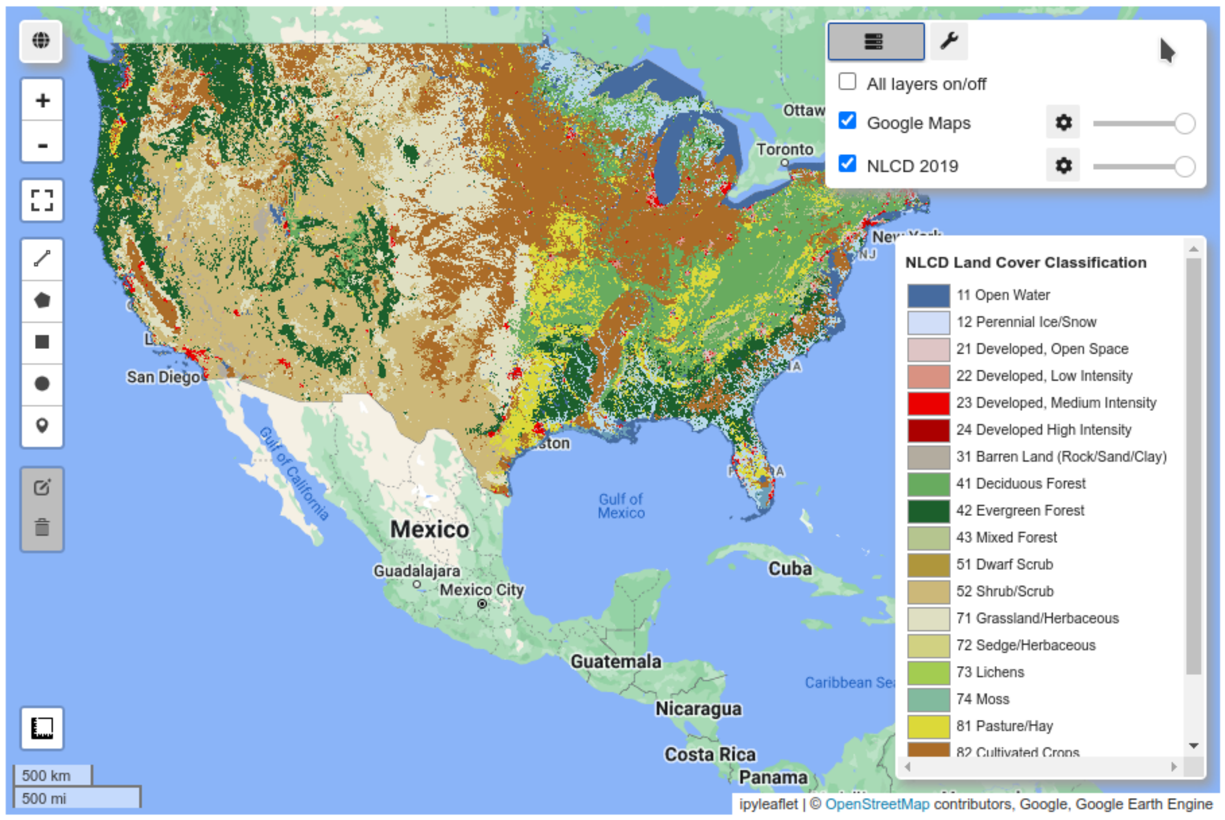


Fig. F6.3.7 The NLCD 2019 image layer displayed in *geemap*

The **Map** above shows only the NLCD 2019 image. To create an Earth Engine App for visualizing land cover change, we need a stack of NLCD images. Let’s print the list of system IDs of all available NLCD images.

```
dataset.aggregate_array('system:id').getInfo()
```

The output should look like this.

```
['USGS/NLCD_RELEASES/2019_REL/NLCD/2001',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2004',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2006',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2008',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2011',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2013',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2016',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2019']
```

Select the eight NLCD epochs after 2000.

```
years = ['2001', '2004', '2006', '2008', '2011', '2013', '2016',
'2019']
```

Define a function for filtering the NLCD `ImageCollection` by year and select the `'landcover'` band.

```
# Get an NLCD image by year.
def getNLCD(year):
    # Import the NLCD collection.
    dataset = ee.ImageCollection('USGS/NLCD_RELEASES/2019_REL/NLCD')

    # Filter the collection by year.
    nlcd = dataset.filter(ee.Filter.eq('system:index', year)).first()

    # Select the land cover band.
    landcover = nlcd.select('landcover');
    return landcover
```

Create an NLCD `ImageCollection` to be used in the split-panel map.

```
# Create an NLCD image collection for the selected years.
collection = ee.ImageCollection(ee.List(years).map(lambda year:
getNLCD(year)))
```

Print out the list of system IDs of the selected NLCD images covering the contiguous United States.

```
collection.aggregate_array('system:id').getInfo()
```

The output should look like this.

```
['USGS/NLCD_RELEASES/2019_REL/NLCD/2001',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2004',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2006',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2008',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2011',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2013',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2016',
 'USGS/NLCD_RELEASES/2019_REL/NLCD/2019']
```

Next, create a list of labels to populate the dropdown list.

```
labels = [f'NLCD {year}' for year in years]
labels
```

The output should look like this.

```
['NLCD 2001',
 'NLCD 2004',
 'NLCD 2006',
 'NLCD 2008',
 'NLCD 2011',
 'NLCD 2013',
 'NLCD 2016',
 'NLCD 2019']
```

The last step is to create a split-panel map by passing the NLCD `ImageCollection` and list of labels to `Map.ts_inspector`.

```
Map.ts_inspector(left_ts=collection, right_ts=collection,
left_names=labels, right_names=labels)
Map
```

The split-panel map should look like Fig. F6.3.8.

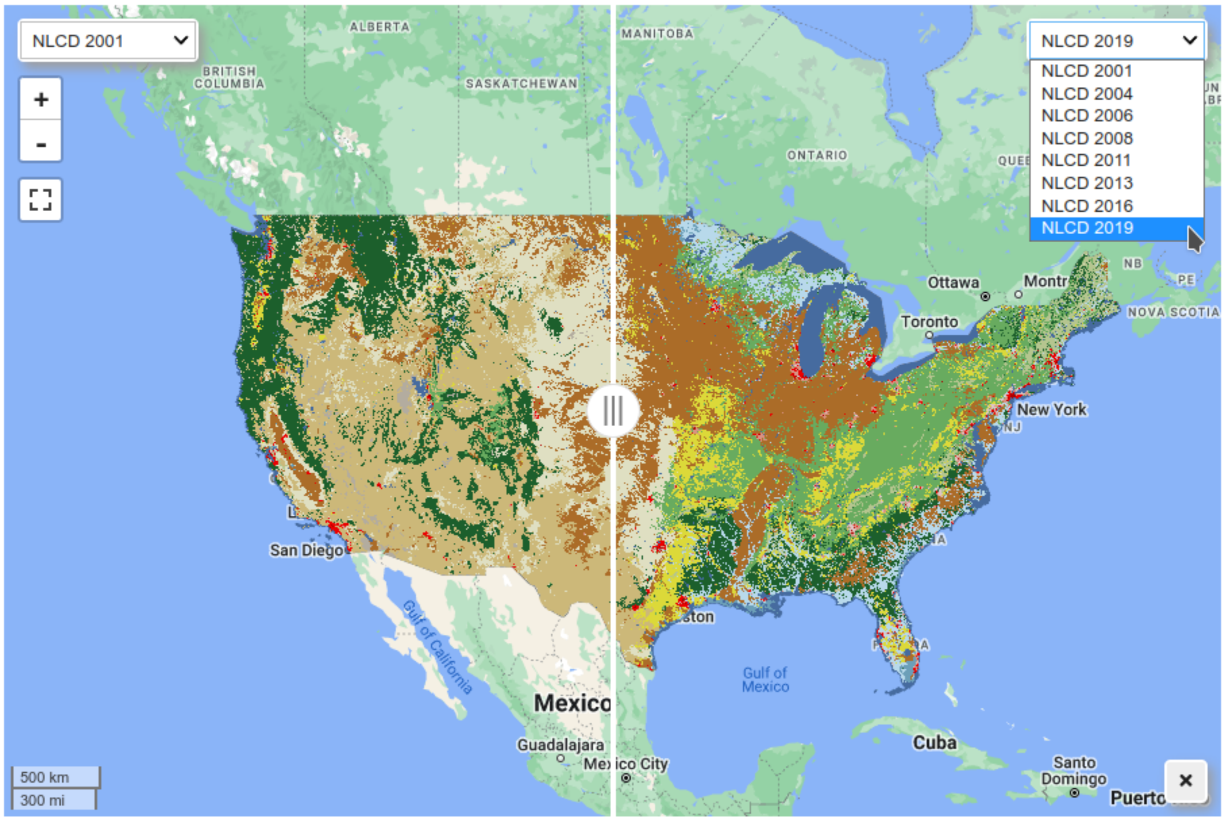


Fig. F6.3.8 A split-panel map for visualizing land cover change with *geemap*

To visualize land cover change, choose one NLCD image from the left dropdown list and another image from the right dropdown list, then use the slider to swipe through to visualize land cover change interactively. Click the **close** button in the bottom-right corner to close the split-panel map and return to the NLCD 2019 image shown in Fig. F6.3.7.

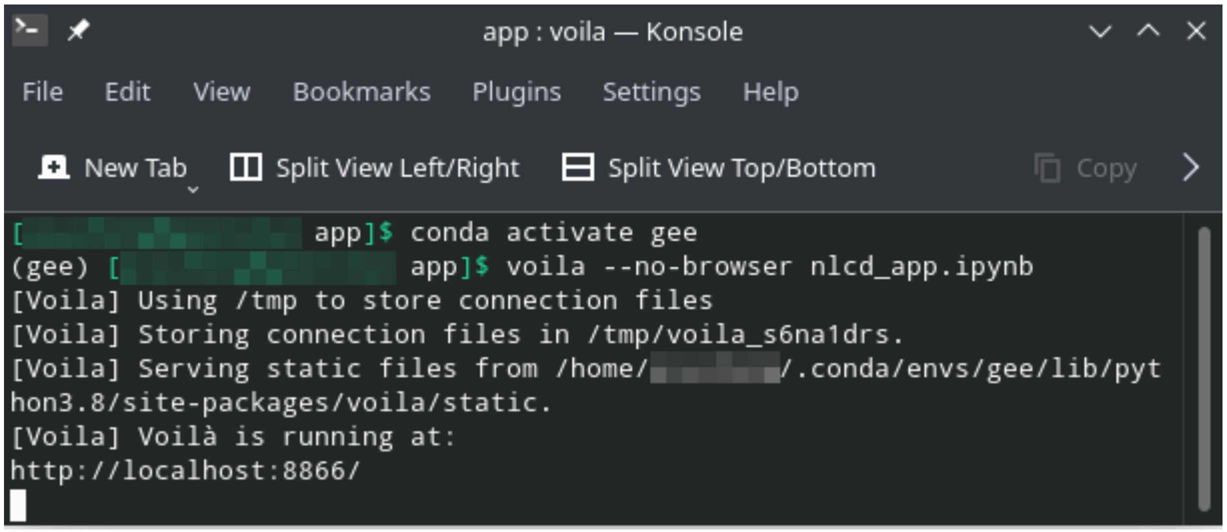
Code Checkpoint F63c. The book's repository contains information about what your code should look like at this point.

Section 4. Publishing an Earth Engine App Using a Local Web Server

In this section, you will learn how to deploy an Earth Engine App using a local computer as a web server. Assume that you have completed Sect. 3 and created a Jupyter notebook named `nlcd_app.ipynb`. First, you need to download *ngrok*, a program that can turn your computer into a secure web server and connect it to the *ngrok* cloud service, which accepts traffic on a public address. Download *ngrok* from https://ngrok.com and unzip it to a directory on your computer, then copy `nlcd_app.ipynb` to the same directory. Open the Anaconda Prompt (on Windows) or the Terminal (on macOS/Linux) and enter the following commands. Make sure you change `/path/to/ngrok/dir` to your computer directory where the *ngrok* executable is located, e.g., `~/Downloads`.

```
cd /path/to/ngrok/dir
conda activate gee
voila --no-browser nlcd_app.ipynb
```

The output of the terminal should look like this.



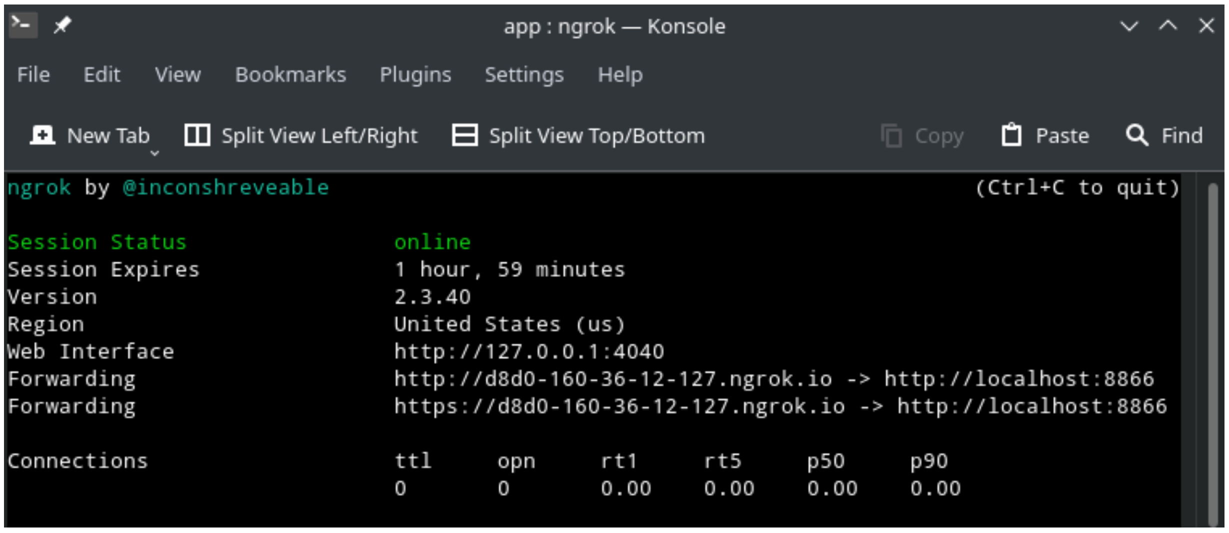
```
[[redacted] app]$ conda activate gee
(gee) [[redacted] app]$ voila --no-browser nlcd_app.ipynb
[Voila] Using /tmp to store connection files
[Voila] Storing connection files in /tmp/voila_s6na1drs.
[Voila] Serving static files from /home/[redacted]/.conda/envs/gee/lib/pyt
hon3.8/site-packages/voila/static.
[Voila] Voilà is running at:
http://localhost:8866/
```

Fig. F6.3.9 The output of the terminal running Voilà

Voilà can be used to run, convert, and serve a Jupyter notebook as a standalone app. Click the link (e.g., http://localhost:8866) shown in the terminal window to launch the interactive dashboard. Note that the port number is 8866, which is needed in the next step to launch *ngrok*. Open another terminal and enter the following command.

```
cd /path/to/ngrok/dir
ngrok http 8866
```

The output of the terminal should look like Fig. F6.3.10. Click the link shown in the terminal window to launch the interactive dashboard. The link should look like https://random-string.ngrok.io, which is publicly accessible. Anyone with the link will be able to launch the interactive dashboard and use the split-panel map to visualize NLCD land cover change. Keep in mind that the dashboard might take several seconds to load, so please be patient.



```
ngrok by @inconshreveable                                          (Ctrl+C to quit)

Session Status                online
Session Expires               1 hour, 59 minutes
Version                       2.3.40
Region                        United States (us)
Web Interface                 http://127.0.0.1:4040
Forwarding                    http://d8d0-160-36-12-127.ngrok.io -> http://localhost:8866
Forwarding                    https://d8d0-160-36-12-127.ngrok.io -> http://localhost:8866

Connections                   ttl     opn     rt1     rt5     p50     p90
                              0       0       0.00    0.00    0.00    0.00
```

Fig. F6.3.10 The output of the terminal running *ngrok*

To stop the web server, press Ctrl + C on both terminal windows. See below for some optional settings for running Voilà and *ngrok*.

To show code cells from your App, run the following from the terminal.

```
voila --no-browser --strip_sources=False nlcd_app.ipynb
```

To protect your App with a password, run the following.

```
ngrok http -auth="username:password" 8866
```

Section 5. Publish an Earth Engine App Using Cloud Platforms

In this section, you will learn how to deploy an Earth Engine App on cloud platforms, such as Heroku and Google App Engine. Heroku is a “platform as a service” that enables developers to build, run, and operate applications entirely in the cloud. It has a free tier with limited computing hours, which would be sufficient for this lab. Follow the steps below to deploy the Earth Engine App on Heroku.

First, go to https://github.com/signup to sign up for a GitHub account if you don’t have one already. Once your GitHub account has been created, log into your account and navigate to the sample app repository: https://github.com/giswqs/earthengine-apps. Click the **Fork** button in the top-right corner to fork this repository into your account. Two important files in the repository are worth mentioning here: requirements.txt lists the required packages (e.g., *geemap*) to run the App, while Procfile specifies the commands that are executed by the App on startup. The content of Procfile should look like this.

```
web: voila --port=$PORT --no-browser --strip_sources=True --
enable_nbextensions=True --MappingKernelManager.cull_interval=60 --
MappingKernelManager.cull_idle_timeout=120 notebooks/
```

The above command instructs the server to hide the source code and periodically check for idle kernels—in this example, every 60 seconds—and cull them if they have been idle for more than 120 seconds. This can avoid idle kernels using up the server computing resources. The page served by Voilà will contain a list of any notebooks in the notebooks directory.

Next, go to https://signup.heroku.com to sign up for a Heroku account if you don't have one already. Log into your account and click the **New** button in the top-right corner, then choose **Create new app** from the dropdown list (Fig. F6.3.11).

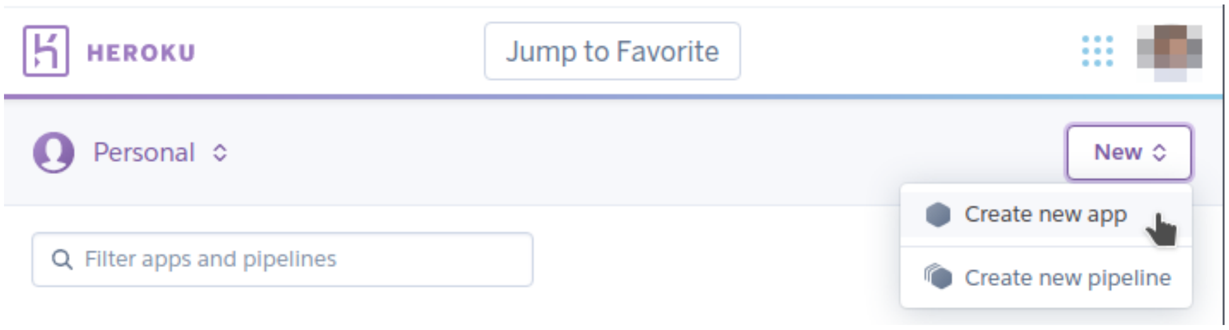
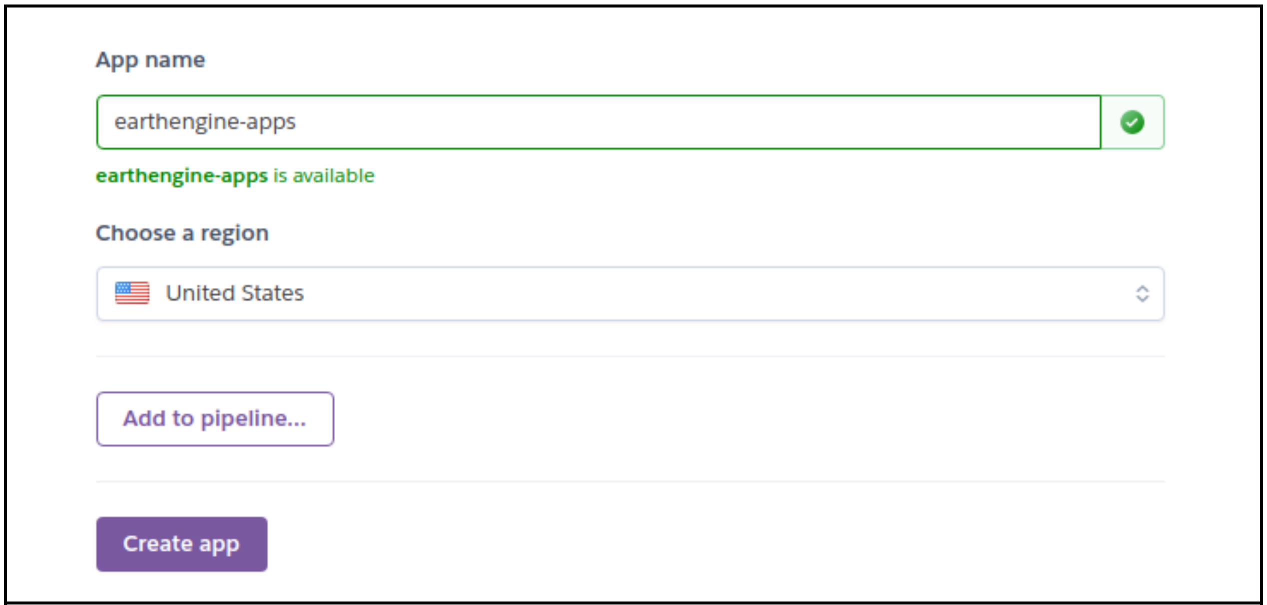


Fig. F6.3.11 Creating a new app in Heroku

Choose a name for your App (Fig. F6.3.12). Note that the App name must be unique; if an App name has already been taken, you won't be able to use it.



App name

earthengine-apps

earthengine-apps is available

Choose a region

United States

Fig. F6.3.12 Choosing an App name

Once the Heroku App has been created, click the **Deploy** tab and choose **GitHub** as the deployment method. Connect to your GitHub account and enter `earthengine-apps` in the search box. The repository should be listed beneath the search box. Click the **Connect** button to connect the repository to Heroku (Fig. F6.3.13).

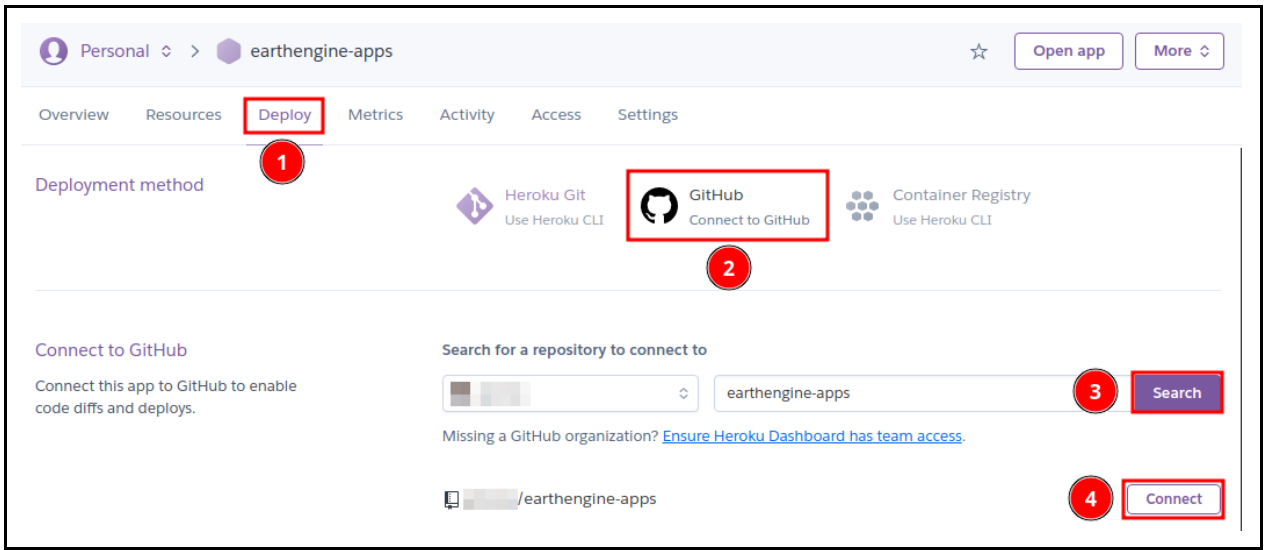
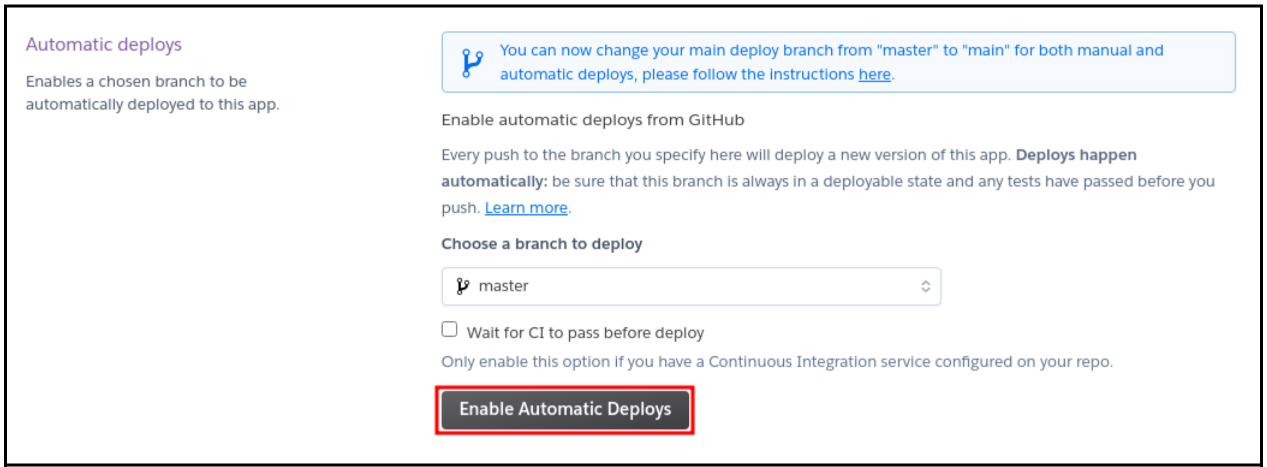


Fig. F6.3.13 Connecting a GitHub account to Heroku

Under the same **Deploy** tab, scroll down and click **Enable Automatic Deploys** (Fig. F6.3.14). This will enable Heroku to deploy a new version of the App whenever the GitHub repository gets updated.


Automatic deploys

Enables a chosen branch to be automatically deployed to this app.

You can now change your main deploy branch from "master" to "main" for both manual and automatic deploys, please follow the instructions here.

Enable automatic deploys from GitHub

Every push to the branch you specify here will deploy a new version of this app. **Deploys happen automatically:** be sure that this branch is always in a deployable state and any tests have passed before you push. Learn more.

Choose a branch to deploy

master

☐ Wait for CI to pass before deploy

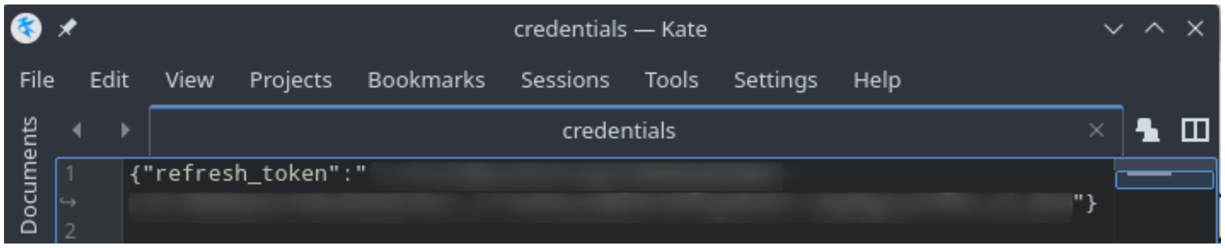
Only enable this option if you have a Continuous Integration service configured on your repo.

Fig. F6.3.14 Enabling Automatic Deploys on Heroku

Since using Earth Engine requires authentication, we need to set the Earth Engine token as an environment variable so that the web App can pass the Earth Engine authentication. If you have completed Sect. 3, you have successfully authenticated Earth Engine on your computer and the token can be found in the following file path, depending on the operating system you are using. Note that you might need to show the hidden directories on your computer in order to see the `.config` folder under the home directory.

```
Windows: C:\Users\USERNAME\.config\earthengine\credentials
Linux:  /home/USERNAME/.config/earthengine/credentials
MacOS:  /Users/USERNAME/.config/earthengine/credentials
```

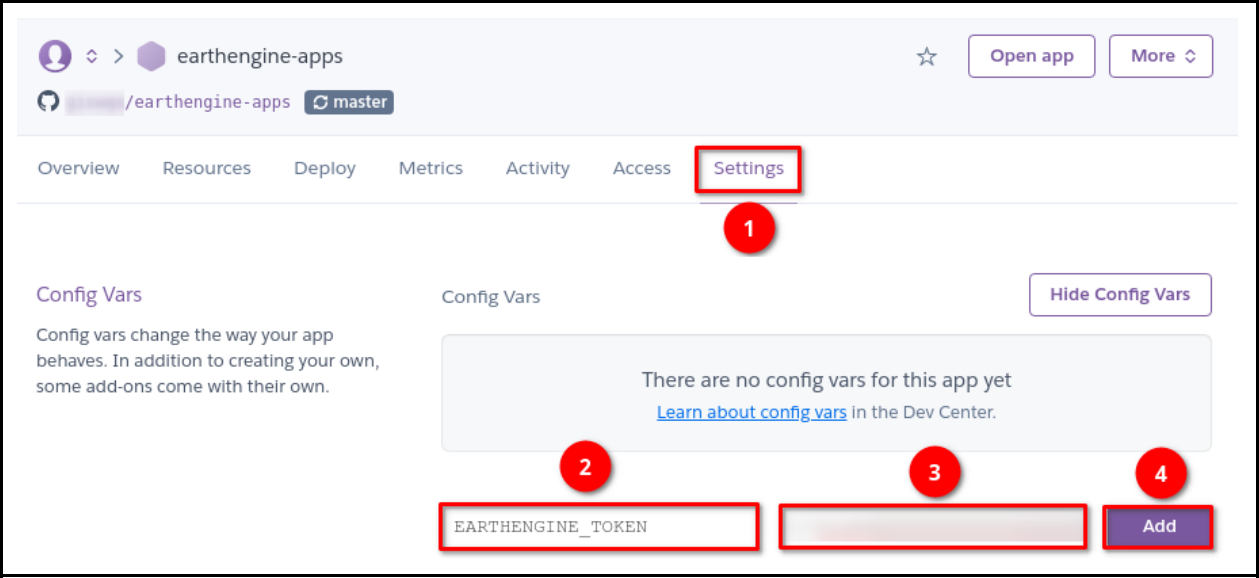
Open the **credentials** file using a text editor. Select and copy the long string wrapped by the double quotes (Fig. F6.3.15). Do not include the double quotes in the copied string.


credentials — Kate
File Edit View Projects Bookmarks Sessions Tools Settings Help
credentials
Documents

```
{"refresh_token":"[redacted]
[redacted]"}
```

Fig. F6.3.15 The Earth Engine authentication token

Next, navigate to your web App on Heroku. Click the **Settings** tab, then click **Reveal Config Vars** on the page. Enter `EARTHENGINE_TOKEN` as the key and paste the string copied above as the value. Click the **Add** button to set `EARTHENGINE_TOKEN` as an environment variable (Fig. F6.3.16) that will be used by *geemap* to authenticate Earth Engine.



Config Vars

Config vars change the way your app behaves. In addition to creating your own, some add-ons come with their own.

Config Vars

There are no config vars for this app yet

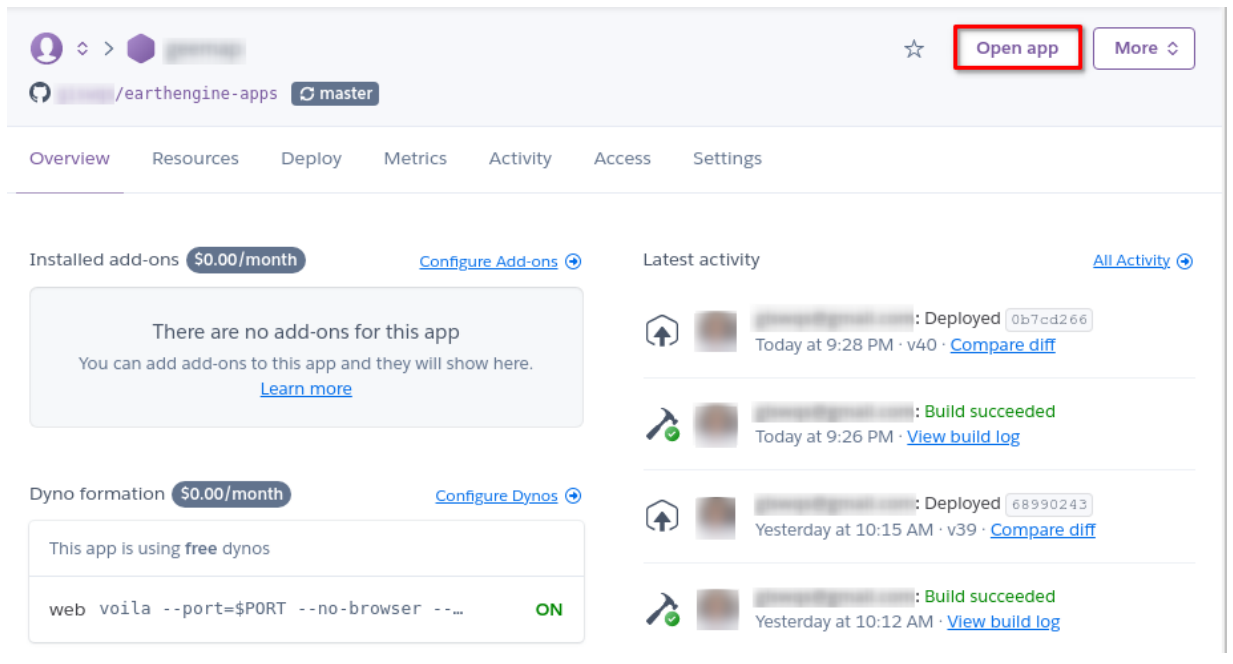
Learn about config vars in the Dev Center.

EARTHENGINE_TOKEN

Fig. F6.3.16 Setting the Earth Engine token as an environment variable on Heroku

The last step is to commit some changes to your forked GitHub repository to trigger Heroku to build and deploy the App. If you are familiar with Git commands, you can push changes to the repository from your local computer to GitHub. If you have not used Git before, you can navigate to your repository on GitHub and make changes directly using the browser. For example, you can navigate to README.md and click the Edit icon on the page to start editing the file. Simply place the cursor at the end of the file and press Enter to add an empty line to the file, then click the **Commit changes** button at the bottom of the page to save changes. This should trigger Heroku to build the App. Check the build status under the latest activities of the **Overview** tab. Once the App has been built and deployed successfully, you can click the **Open app** button in the top-right corner to launch the web App (Fig. F6.3.17). When the App is open in your browser, click `nlcd_app.ipynb` to launch the split-panel map for visualizing land cover change. This is the same notebook that we developed in Sect. 3. The App URL should look like this: https://APP-NAME.herokuapp.com/voila/render/nlcd_app.ipynb.

Congratulations! You have successfully deployed the Earth Engine App on Heroku.



[redacted]
[redacted]/earthengine-apps master

Installed add-ons $0.00/month

There are no add-ons for this app
You can add add-ons to this app and they will show here.
Learn more

Dyno formation $0.00/month

This app is using **free** dynos

```
web  voila --port=$PORT --no-browser --…   ON
```

Latest activity

[redacted]: Deployed 0b7cd266
Today at 9:28 PM · v40 · Compare diff

[redacted]: Build succeeded
Today at 9:26 PM · View build log

[redacted]: Deployed 68990243
Yesterday at 10:15 AM · v39 · Compare diff

[redacted]: Build succeeded
Yesterday at 10:12 AM · View build log

Fig. F6.3.17 Setting the Earth Engine token as an environment variable on Heroku

Code Checkpoint F63d. The book's repository contains information about what your code should look like at this point.

Question 1. What are the pros and cons of designing Earth Engine Apps using *geemap* and *ipywidgets*, compared to the JavaScript Earth Engine User Interface API?

Question 2. What are the pros and cons of deploying Earth Engine Apps on Heroku, compared to a local web server and the Earth Engine Code Editor?

Synthesis

Assignment 1. Replace the NLCD datasets with other multitemporal land cover datasets (e.g., United States Department of Agriculture National Agricultural Statistics Service Cropland Data Layers, visible in the Earth Engine Data Catalog) and modify the web App for visualizing land cover change using the chosen land cover datasets. Deploy the web App using multiple platforms (i.e., JavaScript Code Editor, *ngrok*, and Heroku). More land cover datasets can be found at https://developers.google.com/earth-engine/datasets/tags/landcover.

Conclusion

In this chapter, you learned how to design Earth Engine Apps using both the Earth Engine User Interface API (JavaScript) and *geemap* (Python). You also learned how to deploy Earth Engine Apps on multiple platforms, such as the JavaScript Code Editor, a local web server, and Heroku. The skill of designing and deploying interactive Earth Engine Apps is essential for making your research and data products more accessible to the scientific community and the general public. Anyone with the link to your web App can analyze and visualize Earth Engine datasets without needing an Earth Engine account.

Feedback

To review this chapter and make suggestions or note any problems, please go now to bit.ly/EEFA-review. You can find summary statistics from past reviews at bit.ly/EEFA-reviews-stats.

References

- Earth Engine User Interface API: https://developers.google.com/earth-engine/guides/ui
- Earth Engine Apps: https://developers.google.com/earth-engine/guides/apps
- Voilà: https://voila.readthedocs.io
- *geemap*: https://geemap.org
- *ngrok*: https://ngrok.com
- Heroku: https://heroku.com
- Earthengine-apps: https://github.com/giswqs/earthengine-apps