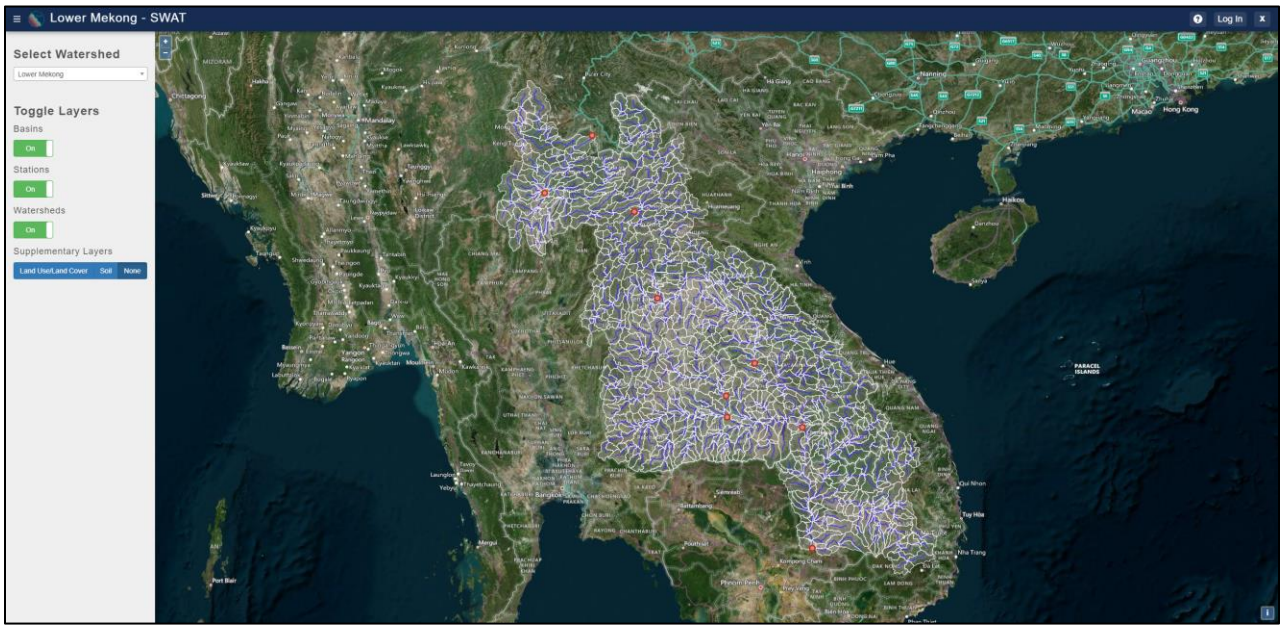
SWAT Data Viewer - Installation Guide



Overview

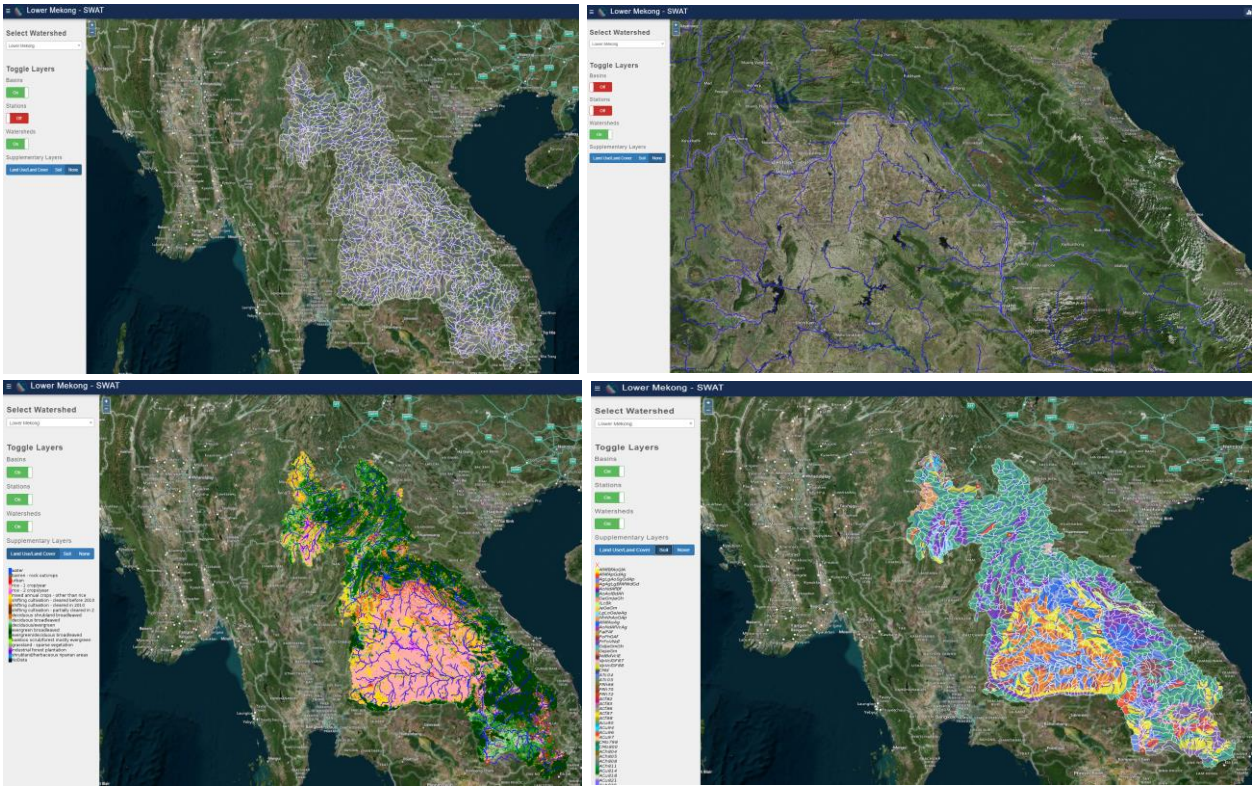
The SWAT Data Viewer was developed to act as a virtual and open access data store for stakeholders and decision-makers to view and download SWAT model inputs and outputs for their area of interest. The app features various spatial and temporal data visualization interfaces (mainly in the form of maps and timeseries plots). Unlike many of the SWAT related web applications, this application is completely modular meaning that it can be duplicated, customized and served from any server running the Tethys platform and can display the data from any valid SWAT model.

This tutorial will outline the various steps required to set up the app on your own server/computer.

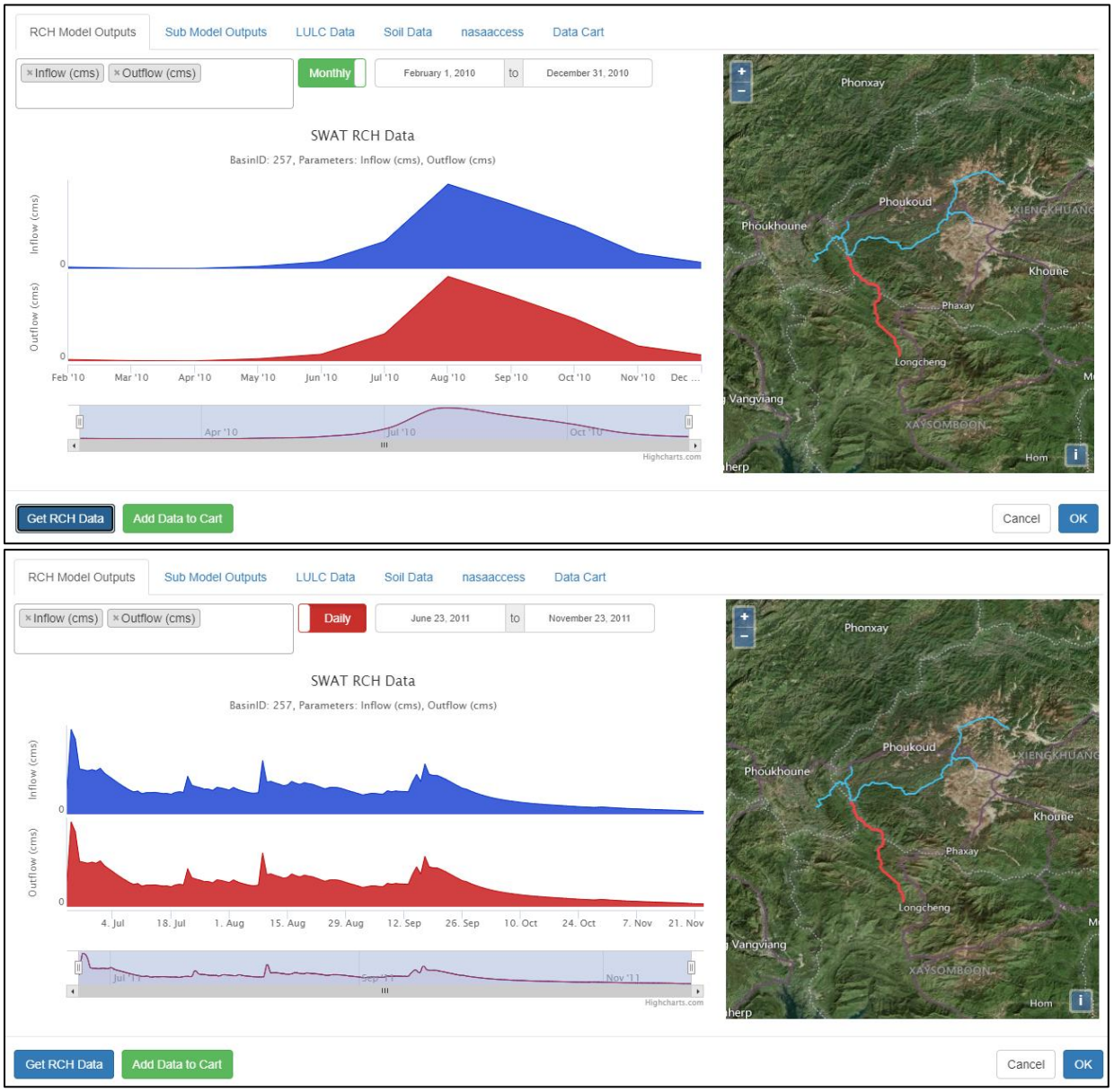
Key features

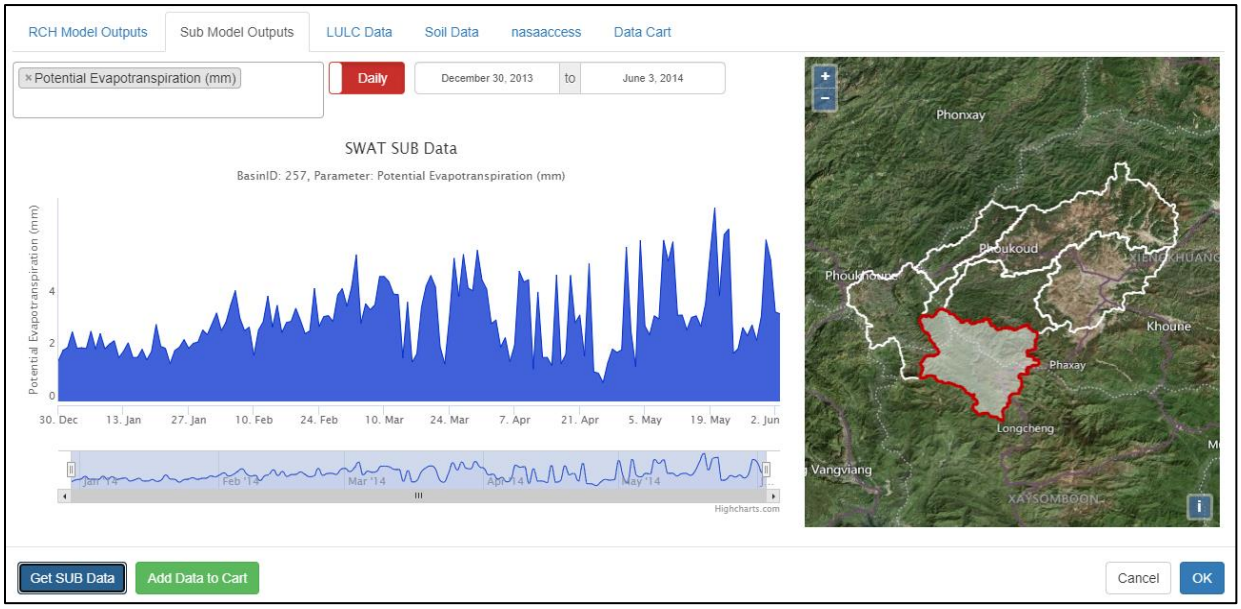
Simply put, the SWAT Data Viewer acts as a data visualization and access portal for any SWAT model that has been uploaded to the app. The following images outline the various features included within the app.

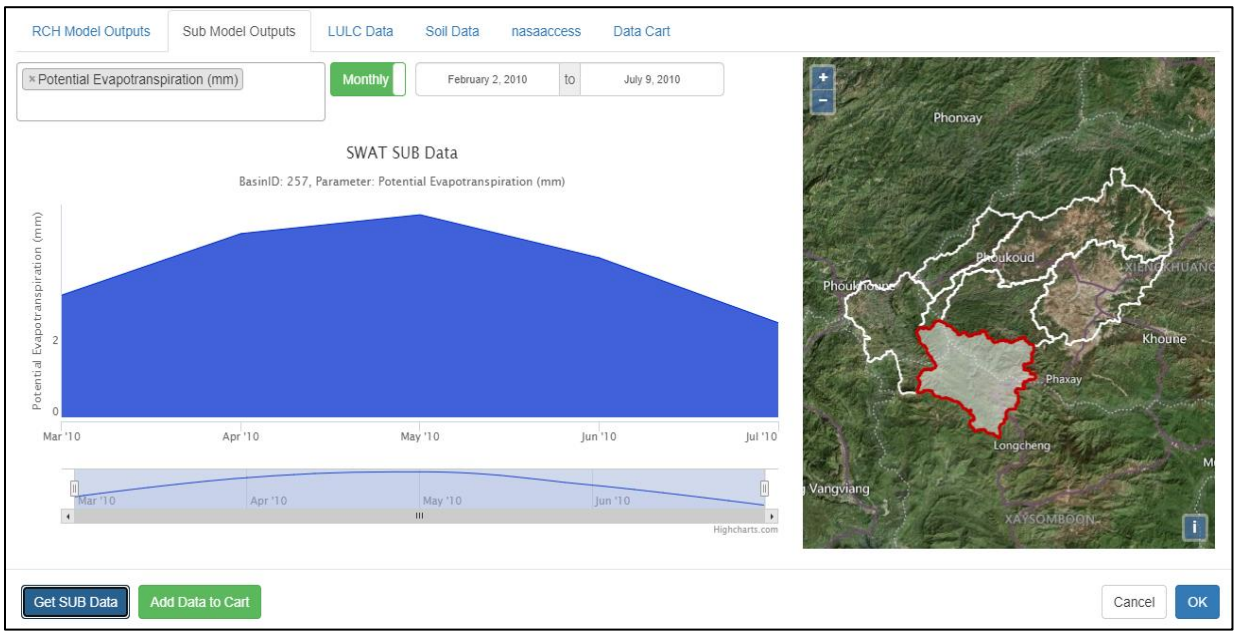
Map visualization of key watershed features



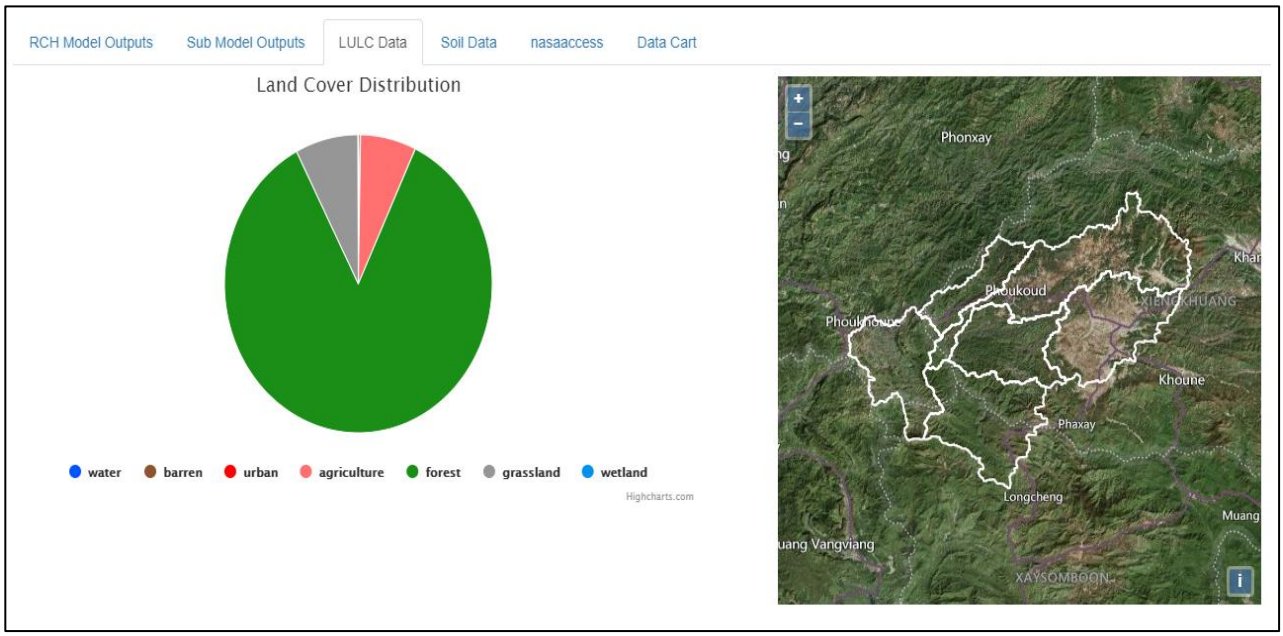
Time series visualization of SWAT outputs

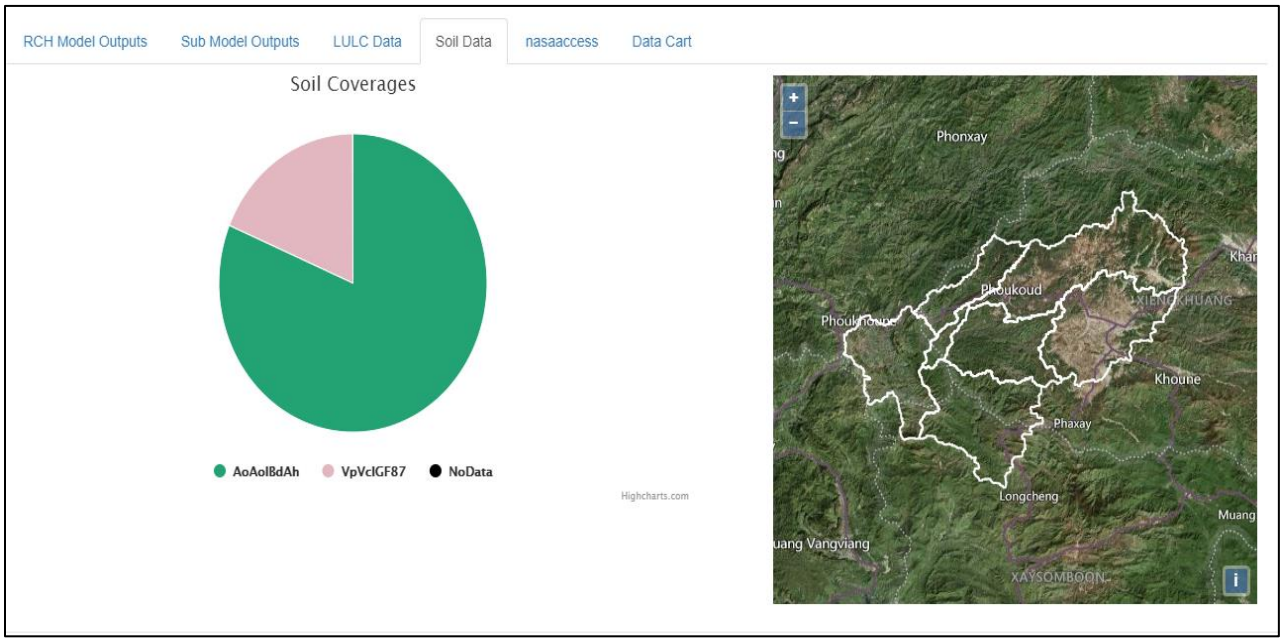



RCH Model Outputs
Sub Model Outputs
LULC Data
Soil Data
nasaaccess
Data Cart
× Potential Evapotranspiration (mm)
Daily
December 30, 2013
to
June 3, 2014
SWAT SUB Data
BasinID: 257, Parameter: Potential Evapotranspiration (mm)
Potential Evapotranspiration (mm)
30. Dec
13. Jan
27. Jan
10. Feb
24. Feb
10. Mar
24. Mar
7. Apr
21. Apr
5. May
19. May
2. Jun
Jan '14
Feb '14
Mar '14
Apr '14
May '14
Highcharts.com
Phonxay
Phoukoud
Phoukhoune
Khoune
Phaxay
Longcheng
Vangviang
XAYSOMBOON
Get SUB Data
Add Data to Cart
Cancel
OK

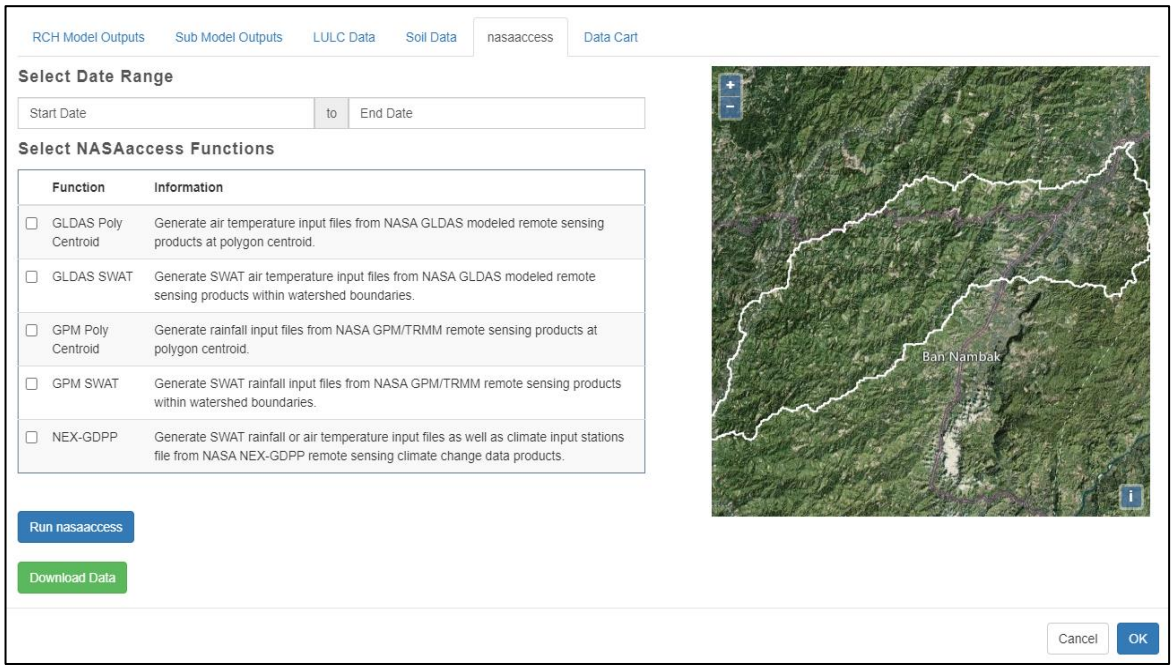

RCH Model Outputs
Sub Model Outputs
LULC Data
Soil Data
nasaaccess
Data Cart
× Potential Evapotranspiration (mm)
Monthly
February 2, 2010
to
July 9, 2010
SWAT SUB Data
BasinID: 257, Parameter: Potential Evapotranspiration (mm)
Potential Evapotranspiration (mm)
Mar '10
Apr '10
May '10
Jun '10
Jul '10
Highcharts.com
Phonxay
Phoukoud
Phoukhoune
Khoune
Phaxay
Longcheng
Vangviang
XAYSOMBOON
Get SUB Data
Add Data to Cart
Cancel
OK

Land Use/Land Cover and Soil coverage statistics

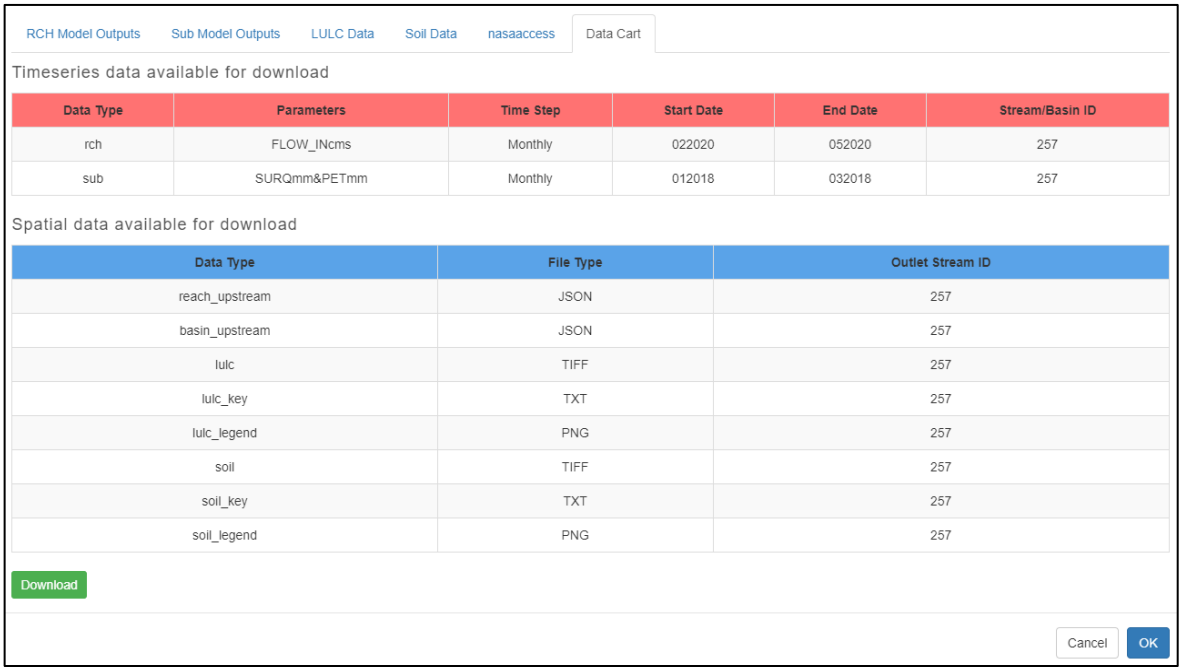




nasaaccess interface



Data cart for downloading data



Installation/Setup Instructions

1. Clone app

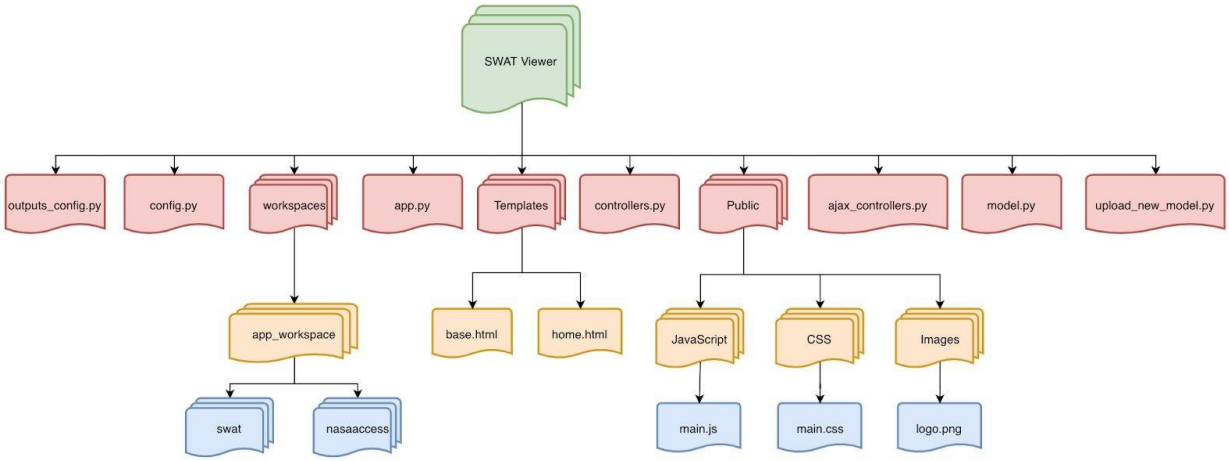
Within the 'tethys' python environment, clone the SWAT Data Viewer git repository to the folder you have decided to use for storing the app packages. If the apps will be installed on a production server, this folder should be located at */home_directory/tethys/apps/.* Use the following command to clone the app.

```
git clone https://github.com/SERVIR/SWAT2.0.git
```

If you plan to make edits to the app and would like to track your own development within the app, forking the repository to your own github account is recommended.

The app package

After cloning the app onto the server/computer, the following file structure should be created in the tethysapp-swat2 folder.



2. Install prerequisites

This app could be installed in Windows and Linux based operating systems.

While still in the 'tethys' environment, install the following packages using the command below:

```
conda install -c conda-forge package_name
```

numpy=1.11.3
pandas=0.23.4
requests=2.18.4
dbfread=2.0.7
gdal=2.0.0
It is possible that other versions of these packages will work, but the versions specified above have been tested and have proven to work.

3. Setup nasaaccess

Install R>=4.0.5 version from https://cran.r-project.org/. Follow the instructions here https://github.com/nasa/NASAaccess/ and set up the nasaaccess R package.

Data requirements and file structure

The SWAT web app uses two different folders to store and write data to: (1) the nasaaccess_data folder in the server and the user_workspace folder in the app package.

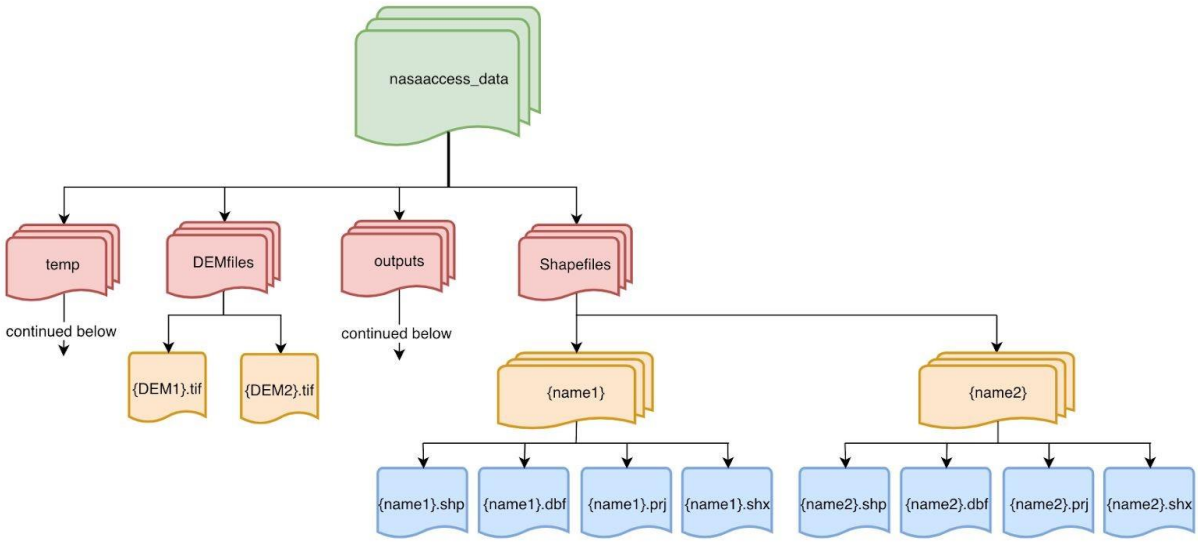
nasaaccess_data folder

The nasaaccess_data folder is where the app will look to find the input shapefile, DEM file, and output folder to write the outputs to.

Within nasaaccess_data, there are four subfolders:

1) DEMfiles
    a) Contains all the tif files that the app admin wants to make available for all users of the app
    b) These files need to be in a geographic coordinate system for the functions to work
2) Shapefiles
    a) Contains all the shapefiles that the app admin wants to make available for all users of the app
    b) Each watershed/boundary is given it's own subfolder in the Shapefiles folder that contains the 4 separate shapefile component files (shp, shx, dbf, and prj).
        i) the name of this folder is what will show up in the user interface dropdown and should be the same name used for each of the 4 files
    c) These files need to be in a geographic coordinate system for the functions to work
3) Outputs
    a) This is the folder that the app will write the function outputs to.
    b) Each new user request from the app, creates a new uniquely name subfolder within the outputs folder and the outputs are written there
4) Temp
    a) This is where all of the intermediate files (i.e. raw data netCDF files downloaded from EarthData) are stored.

b) These files will be deleted when the nasaaccess functions have completed so this folder should remain empty unless a process is running



4. Deletion script for nasaaccess data

Make sure you have a script set up in the crontab that deletes the old nasaaccess data.
Following the script that will be required for the crontab

```
import os, time, shutil

path = r"/home_directory/nasaaccess_data/outputs"
now = time.time()

for filename in os.listdir(path):
    # if os.stat(os.path.join(path, filename)).st_mtime < now - 7 * 86400:
    if os.path.getmtime(os.path.join(path, filename)) < now - 5 * 86400:
        if os.path.isdir(os.path.join(path, filename)):
            print(filename)
            shutil.rmtree(os.path.join(path, filename))
```

5. Obtain requisite data and set up file structure

For the SWAT Data Viewer application to function fully, there needs to be at least one SWAT model available for the app to view. All of the data/files that the app uses will be located in one of four locations: (1) File on the server, (2) PostgreSQL database, (3) Geoserver, and (4) the

app's workspace (app_workspace folder in the app package). The table below describes the various files/data that the app is able to process, where it needs to be located, and whether the file is required for the app to run.

Data/File	Description	Locations	Req'd?
Stream Shapefile (.zip file containing .shp, .shx, .dbf, and .prj files)	Polyline shapefile containing stream reach information. Required attribute fields: "subbasin", "to_node"	Geoserver PostgreSQL	Yes
Subbasin Shapefile (.zip file containing .shp, .shx, .dbf, and .prj files)	Polygon shapefile containing subbasin information Required attribute fields: "subbasin"	Geoserver	Yes
Gauge stations Shapefile (.zip file containing .shp, .shx, .dbf, and .prj files)	Point shapefile containing gauge station information	Geoserver PostgreSQL	No
output.rch and output.sub	Daily SWAT model output files with IDs that correspond to the "subbasin" fields in the stream and subbasin shapefiles	PostgreSQL	At least one of them
Land use/Land cover (lulc) map with lookup table	TIFF file containing lulc information over the watershed Comma delimited .txt file containing lulc classes (i.e agriculture), subclasses (i.e. rice crops), and hexidecimal color codes for each value in the lulc TIFF file	Geoserver PostgreSQL File on server	No
Soil map with lookup table	TIFF file containing soil information over the watershed Comma delimited .txt file contatining soil type names and a hexidecimal color code for each value in the soil TIFF file	Geoserver PostgreSQL File on server	No
DEM	TIFF file containing elevation data over the watershed to be used when users want to run nasaaccess functions	File on server	No
	from the SWAT data viewer	PostgreSQL	

The initial installation of the SWAT Data Viewer does not require any of these files to be on the server but instead will be used to initialize a new empty PostgreSQL database (See sections below for descriptions of each individual table in the database.

The swat_db PostgreSQL database

The app needs a PostgreSQL database. Please create an instance if you do not have one setup already. Once the database is available, create the required tables in the following format. The following sections will outline the various tables in the swat_db PostgreSQL. These tables are used by the app to visualize and process data. Please use the following scripts to create the tables:

watershed

```
CREATE TABLE watershed
(
    id integer NOT NULL DEFAULT nextval('watershed_id_seq'::regclass),
    name character varying COLLATE pg_catalog."default",
    CONSTRAINT watershed_pkey PRIMARY KEY (id)
)
```

Field	Description
id	ID number that will be used throughout the other tables to ensure the app knows what data belongs to which watershed
name	Name of the watershed/SWAT model This will be the same name used for all files related to the given watershed/model

output_rch

```
CREATE TABLE output_rch
(
    id integer NOT NULL DEFAULT nextval('output_rch_id_seq'::regclass),
    watershed_id integer,
    year_month_day date,
    reach_id integer,
    var_name character varying COLLATE pg_catalog."default",
    val double precision,
    CONSTRAINT output_rch_pkey PRIMARY KEY (id),
    CONSTRAINT output_rch_watershed_id_fkey FOREIGN KEY (watershed_id)
        REFERENCES watershed (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

Field	Description
watershed_id	ID number corresponding the the id field in the “watershed” table
year_month_day	The date that the value in the ‘val’ field represents

reach_id	ID of the stream reach within the watershed Should match the ‘subbasin’ field in the stream shapefile
var_name	Variable name that the value in the ‘val’ field represents
val	Data value for a given reach_id, date, and variable combination

output_sub

```
CREATE TABLE output_sub
(
    id integer NOT NULL DEFAULT nextval('output_sub_id_seq'::regclass),
    watershed_id integer,
    year_month_day date,
    sub_id integer,
    var_name character varying COLLATE pg_catalog."default",
    val double precision,
    CONSTRAINT output_sub_pkey PRIMARY KEY (id),
    CONSTRAINT output_sub_watershed_id_fkey FOREIGN KEY (watershed_id)
        REFERENCES watershed (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

Field	**Description**
watershed_id	ID number corresponding the the id field in the “watershed” table
year_month_day	The date that the value in the ‘val’ field represents
sub_id	ID of the subbasin within the watershed Should match the ‘subbasin’ field in the Subbasin shapefile
var_name	Variable name that the value in the ‘val’ field represents
val	Data value for a given subbasin_id, date, and variable combination

watershed_info

```
CREATE TABLE watershed_info
(
    id integer NOT NULL DEFAULT nextval('watershed_info_id_seq'::regclass),
```

```
    watershed_id integer,
    rch_start date,
    rch_end date,
    rch_vars character varying COLLATE pg_catalog."default",
    sub_start date,
    sub_end date,
    sub_vars character varying COLLATE pg_catalog."default",
    lulc character varying COLLATE pg_catalog."default",
    soil character varying COLLATE pg_catalog."default",
    stations character varying COLLATE pg_catalog."default",
    rch character varying COLLATE pg_catalog."default",
    sub character varying COLLATE pg_catalog."default",
    nasaaccess character varying COLLATE pg_catalog."default",
    CONSTRAINT watershed_info_pkey PRIMARY KEY (id),
    CONSTRAINT watershed_info_watershed_id_fkey FOREIGN KEY (watershed_id)
      REFERENCES watershed (id) MATCH SIMPLE
      ON UPDATE NO ACTION
      ON DELETE NO ACTION
)
```

Field	**Description**
watershed_id	ID number corresponding the the id field in the “watershed” table
rch_start	First date that rch data is available for a given watershed (obtained from output_rch table)
rch_end	Last date that rch data is available for a given watershed (obtained from output_rch table)
rch_vars	Available rch variables for a given watershed (obtained from output_rch table)
sub_start	First date that sub data is available for a given watershed (obtained from output_sub table)
sub_end	Last date that sub data is available for a given watershed (obtained from output_sub table)
sub_vars	Available sub variables for a given watershed (obtained from output_sub table)

lulc	Was there a lulc map provided for the given watershed? (‘Yes’/’No’)
soil	Was there a soil map provided for the given watershed? (‘Yes’/’No’)
stations	Was there a gauge stations shapefile provided for the given watershed? (‘Yes’/’No’)
rch	Was there an output.rch file provided for the given watershed? (‘Yes’/’No’)
sub	Was there an output.sub file provided for the given watershed? (‘Yes’/’No’)
nasaaccess	Was there a DEM file provided for the given watershed to use for nasaaccess processing? (‘Yes’/’No’)

lulc

```
CREATE TABLE lulc
(
    id integer NOT NULL DEFAULT nextval('lulc_id_seq'::regclass),
    watershed_id integer,
    value integer,
    lulc character varying COLLATE pg_catalog."default",
    lulc_class character varying COLLATE pg_catalog."default",
    lulc_subclass character varying COLLATE pg_catalog."default",
    class_color character varying COLLATE pg_catalog."default",
    subclass_color character varying COLLATE pg_catalog."default",
    CONSTRAINT lulc_pkey PRIMARY KEY (id),
    CONSTRAINT lulc_watershed_id_fkey FOREIGN KEY (watershed_id)
        REFERENCES watershed (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

Field	**Description**
watershed_id	ID number corresponding the the id field in the “watershed” table
value	Unique value in the lulc TIFF file that corresponds to a land use/land cover type
lulc	Four character classification of the lulc type (i.e. RICE, WATR)

lulc_class	lulc class name (i.e. forest)
lulc_subclass	Lulc subclass name (i.e. ‘evergreen’ and ‘deciduous’ would be subclasses of the ‘forest’ class)
class_color	Color that the app will use to display a given lulc class’ coverage percentage in a pie chart
subclass_color	Color that the app will use to display a given lulc subclass’ coverage percentage in a pie chart and that will correspond to the display of the lulc map (This color on the pie chart should match the color on the map)

soil

```
CREATE TABLE soil
(
    id integer NOT NULL DEFAULT nextval('soil_id_seq'::regclass),
    watershed_id integer,
    value integer,
    soil_class character varying COLLATE pg_catalog."default",
    class_color character varying COLLATE pg_catalog."default",
    CONSTRAINT soil_pkey PRIMARY KEY (id),
    CONSTRAINT soil_watershed_id_fkey FOREIGN KEY (watershed_id)
        REFERENCES watershed (id) MATCH SIMPLE
        ON UPDATE NO ACTION
        ON DELETE NO ACTION
)
```

Field	**Description**
watershed_id	ID number corresponding the the id field in the “watershed” table
value	Unique value in the soil TIFF file that corresponds to a land use/land cover type
soil_class	Soil class name
class_color	Color that the app will use to display a given lulc class’ coverage percentage in a pie chart and that will correspond to the display of the soil map (This color on the pie chart should match the color on the map)

stream_connect

```
CREATE TABLE stream_connect
(
  id integer NOT NULL DEFAULT nextval('stream_connect_id_seq'::regclass),
  watershed_id integer,
  stream_id integer,
  to_node integer,
  CONSTRAINT stream_connect_pkey PRIMARY KEY (id),
  CONSTRAINT stream_connect_watershed_id_fkey FOREIGN KEY (watershed_id)
    REFERENCES watershed (id) MATCH SIMPLE
    ON UPDATE NO ACTION
    ON DELETE NO ACTION
)
```

Field	Description
watershed_id	ID number corresponding the the id field in the “watershed” table
stream_id	ID of a stream in the stream shapefile
to_node	ID of the stream downstream of the given stream_id

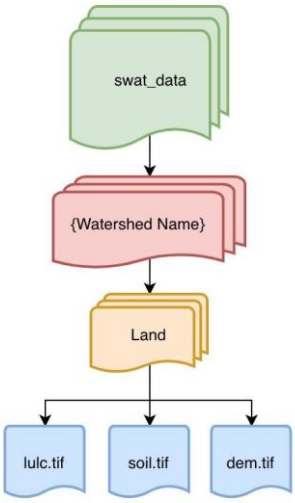
After installing the app, follow the instructions on the “Upload New SWAT Model” tutorial to learn more about each of the file requirements, and to learn how to upload a new watershed to the app. Please make sure there is data uploaded to the database before using the app.

File structure on the server

Of the files/data needed for the app to run, there are only three files that need to remain stored on the server once the data has been uploaded to the geoserver and database: (1) lulc.tif, (2) soil.tif, and (3) dem.tif. These three raster files need to remain on the server because the app runs external processes (detached from the app) on them to clip the lulc and soil files and to run the nasaaccess functions using the DEM file.

Data folder

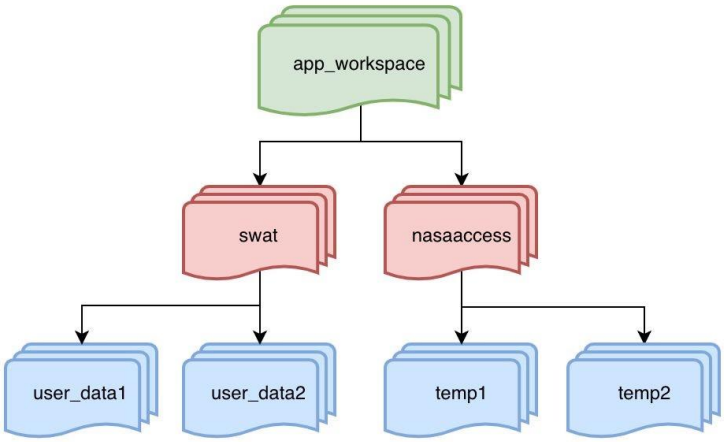
Somewhere on the server, create a folder called “swat_data”. This will be the folder that contains the raster files for each of the watersheds/SWAT models that the app is hosting. The image below shows the file structure that the app will look for (i.e. path/to/swat_data/{watershed_name}/Land/{lulc/soil/dem}.tif).



Note: The swat_data folder can contain data for multiple watersheds/SWAT models.

App workspace

The app_workspace is split into two subfolders: swat and nasaaccess. These two folders are used for temporary file storage and to separate out user data requests into unique “data cart” folders.



swat folder

For each new session in the app, a unique folder is created in the swat folder that will act as the user’s data cart. When the app runs various process (e.g. writing subset stream/subbasin json files and clipping rasters) or when the user requests to add a new timeseries to the “Data Cart” those files are added to this folder. From the “Data Cart” tab, this is the folder that is downloaded when the user elects to download their data.

nasaaccess folder

The nasaaccess functions download and create multiple intermediate files. These files are all temporary but to keep things organized, the app will write these files to the nasaaccess folder in the app_workspace. The files will be deleted when nasaaccess has finished running.

Geoserver

All of the map layers shown in the app are served using Geoserver. The "Upload New SWAT Model" tutorial will outline how to upload data to the geoserver. For this tutorial, make sure you have access to a geoserver (including url, username, and password) or install a new one by following the instructions found here.

6. Reference file paths and geoserver url in config.py and main.js

The swat_data folder can be placed anywhere on the server as long as its path is referenced in the config.py file in the app package as shown below. The config.py is used by the app to locate the various scripts and files that it needs. Please create the config.py file in tethysapp/swat2 directory if it does not exist. It should have the following content.

```
import os
temp_workspace = os.path.join('/home_directory/swat_data', 'swat')
data_path = os.path.join('/home_directory/swat_data')
gdalwarp_path = os.path.join('/home_directory/miniconda/envs/tethys/bin/gdalwarp')
geoserver = {'rest_url':'https://thredds.servirglobal.net/geoserver/rest/',
        'wms_url':'https://thredds.servirglobal.net/geoserver/wms/',
        'wfs_url':'https://thredds.servirglobal.net/geoserver/wfs/',
        'user':<<username>>,
        'password':<<password>>,
        'workspace':'swat'
        }
db = {'name': <<database_name>>,
       'user':<<database_user>>,
       'pass':<<database_password>>,
       'host':<<IP_address>>,
       'port':<<port>>}
R_path = os.path.join('/home_directory/nasaaccess_data')
```

```
R_temp = os.path.join('/home_directory/swat_data', 'nasaaccess')
nasaaccess_R = os.path.join('/..path_to.../Rscript')
R_script = os.path.join('/home_directory/subprocesses/nasaaccess.R')
R_log = os.path.join('/home_directory/subprocesses/nasaaccess_R.log')
```

1) temp_workspace: path to app workspace used for writing temp files and user data cart folders
    a) The app will automatically generate this file path.
2) data_path: path to the top level of the swat_data directory.
    a) If this folder is structured as shown above, the app will know how to access all the data it contains
3) gdalwarp_path: Path to the gdalwarp executable that will be used to clip the lulc and soil rasters
    a) If you're not sure where this file is run the following command in the terminal:

```
find . -name gdalwarp
```

    b) If there are multiple paths listed, use the one within the tethys miniconda environment (miniconda/envs/tethys/…)
4) geoserver: Specify URLs, username, password, and workspace that the app will use to display and upload spatial data
5) R_path: Path to the top level of the nasaaccess_data directory
6) R_temp: Path to nasaaccess directory inside swat_data directory.
7) R_script: Path to the nasaaccess.R script
8) R_log: Path to the nasaaccess.log file
9) nasaaccess_R: Path to Rscript (/home_directory/miniconda/envs/tethys/bin/Rscript)

Wherever you end up copying the nasaaccess.R file to, create a new empty text file called nasaaccess.log. The app will use this file to create a running log of how the app is being used and will assist in debugging if issues arise.

7. Install the app

Once the config.py file has been updated with the new file paths and URLs, the app is ready to be installed.

Local (Development) Installation

Within the tethys environment, navigate the /path/to/app/package/tethysapp-swat2/ and run the following command in the terminal:

```
tethys install -d
```

Production Installation

Within the tethys environment, navigate the /path/to/app/package/tethysapp-swat2/ and run the following commands in the terminal:

```
tuo
Tethys install
tethys manage collectall
tso
tsr
```

8. Set file permissions for the app

Read and write permissions and ownership requirements

In a production installation of the SWAT Data Viewer application, there are various file permission and file ownership changes that need to be made to the server. When a user requests to clip the lulc or soil rasters on the server, the app tries to run the script as the 'www-data' user instead of the the root user. Unless you change the ownership of the data folders and the executable to 'www-data' the server will not allow the process to run. In addition, because the app will be reading and writing new files to the server, full read, write, and execute permissions (rwx) need to be given to the folders containing data relevant to the app.

Below is a list of the directories that need to have ownership and permissions changed:

Folder	Owner (user:group)	rwx
/path/to/swat_data/folder/*	www-data:www-data	777
/path/to/tethys/apps/*	www-data:www-data	775
/path/to/tethys/src/*	www-data:www-data	775

/path/to/tethys/static/*	www-data:www-data	775
/path/to/tethys/miniconda/envs/*	www-data:www-data	775
/path/to/tethys/miniconda/envs/tethys/lib/python2.7/site-packages/tethysapp/*	www-data:www-data	777
/path/to/swat_app_package/workspaces/*	www-data:www-data	777

* Include all files within that folder

note: 777 corresponds to full read, write, execute permissions and 775 corresponds to limiting write permissions for non-owner users

To change the ownership of a folder and all the subfolders and files it contains:

1) Navigate to that folder in the terminal 2)
Run the following command:

```
sudo chown -R www-data:www-data .
```

To change the read-write-execute permissions for a folder and all the subfolders and files it contains:

1) Navigate to that folder in the terminal
2) Run the following command:

```
sudo chmod -R 0777 .
```

9. Initialized swat_db database (if it isn't already)

10. Upload SWAT model files

The instructions for uploading a new SWAT model can be found in the "Upload New SWAT Model" tutorial.