Astral – Manual

Contents

[Introduction 1](#_Toc60501360)

[Overview 1](#_Toc60501361)

[Installation process 2](#_Toc60501362)

[Running 5](#_Toc60501363)

[Prerequisites 5](#_Toc60501364)

[Usage 7](#_Toc60501365)

[Airflow application 7](#_Toc60501366)

[Overview 8](#_Toc60501367)

[Pipeline 9](#_Toc60501368)

[Usage 11](#_Toc60501369)

[Airflow configuration 14](#_Toc60501370)

[Streamlit application 21](#_Toc60501371)

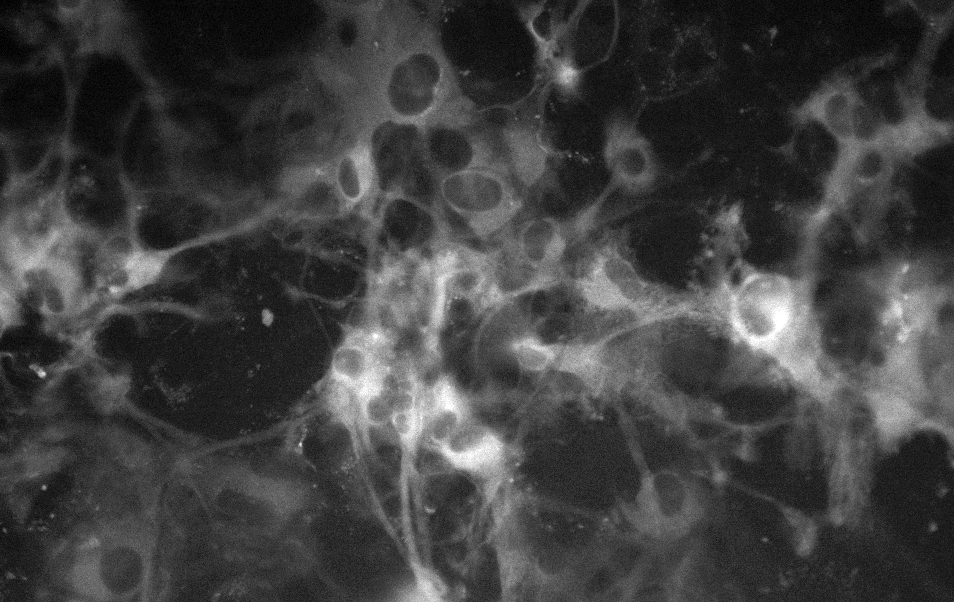
[Overview 21](#_Toc60501372)

[Usage 21](#_Toc60501373)

# Introduction

## Overview

Astral is a tool for analyzing and interpreting calcium signalling from a microscopic timelapse imaging. The goal of Astral is to create an easily extensible system for such data. As such experiments produce complex data, Astral was created for detection, segmentation, and analysis of calcium events inside timelapse data.

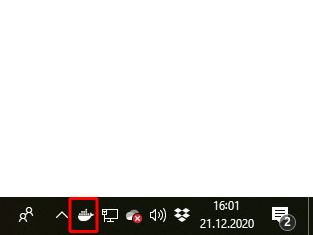


Astral was created in collaboration between Department of Data Science and Engineering of Silesian University of Technology in Gliwice and NeuroScienceLab of Essen University Hospital.

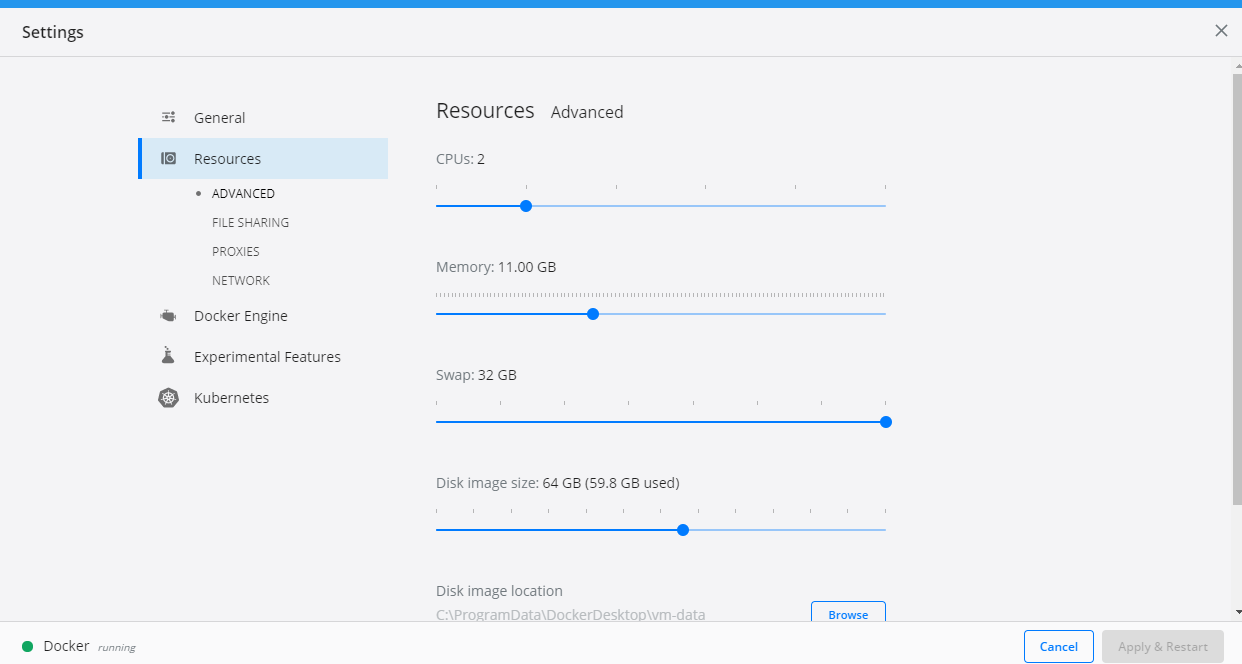
## Installation process

To run Astral, a Docker Desktop application is required. To download Docker Desktop, visit: <https://www.docker.com/products/docker-desktop>

After the installation process, you can launch Docker Desktop, and in the task bar, you should see Docker Desktop’s icon.



Right-click on it and navigate to **Settings**.



In the **Resources** tab, there are specifications for the computational power, that will be dedicated for Docker applications. You may manually customize, how much CPUs, Memory, Swap and Disk size you will give to Docker during runtime.

-------------------------------------------------------------------------------------------------------------------------

Hint

If the Resources tab is not visible in the Docker panel, it is necessary to install Hyper-V service. The dialog window from the Docker Desktop should appear with the steps to install Hyper-V. In other case, visit:

<https://docs.microsoft.com/en-us/virtualization/hyper-v-on-windows/quick-start/enable-hyper-v>

-------------------------------------------------------------------------------------------------------------------------

For Astral to run properly, a significant amount of Virtual Memory is necessary. If possible, set Memory to at least 10GB. For Swap, set it to 32 GB.

Swap is a memory which is used when the application runs out of RAM. Because the timelapse data can be resource-consuming, it is advised to set it as high as possible.

-------------------------------------------------------------------------------------------------------------------------

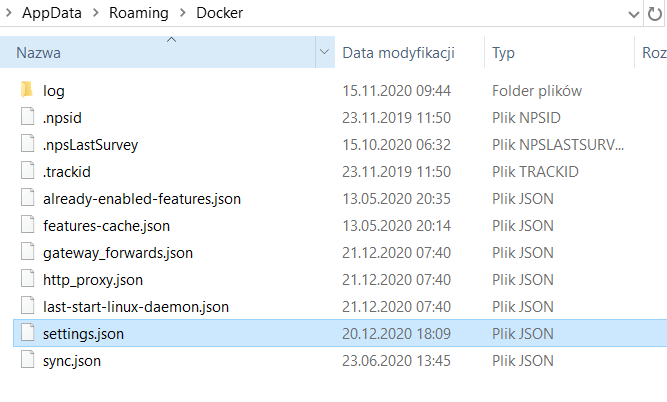
Hint

If you are unable to set Swap to higher values, open up Windows Run window and type:

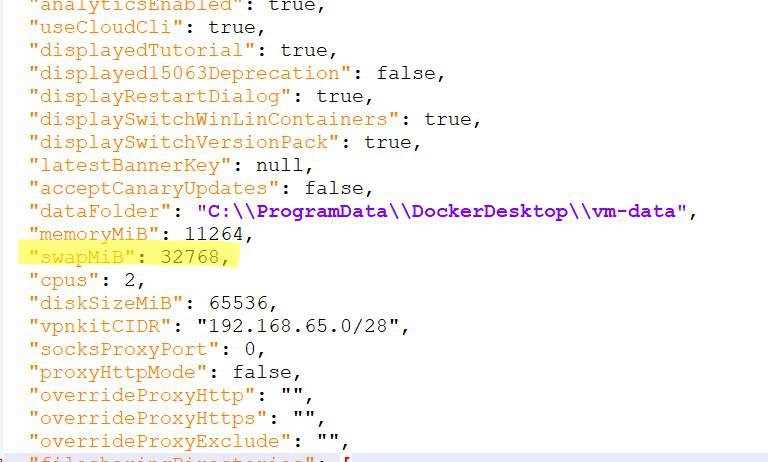
%appdata%

An explorer will show with an AppData folder. Then, navigate to the following file in the explorer.

AppData\Roaming\Docker\setting.json



And open it with your favorite editor. Then, to set the swap memory manually, find the swapMiB property:



And set it to a desired value.

-------------------------------------------------------------------------------------------------------------------------

Depending on assigned RAM memory amount, swap can be set empirically. For image series of 700Mb size and 8 GB of RAM, a good amount for swap is 32 GB, although the amount may be much smaller for some files. The amount of memory is really dependent on the number of events in the image series. This memory is released after application termination, so memory assignment is temporary.

Warning

Remember that swap is used when the program runs out of RAM memory. If the program will need to allocate resources in the swap, but there will not be enough swap memory, the task will crash. In case the step crashes, first thing to check is to increase swap memory and re-run the task.

# Running

## Prerequisites

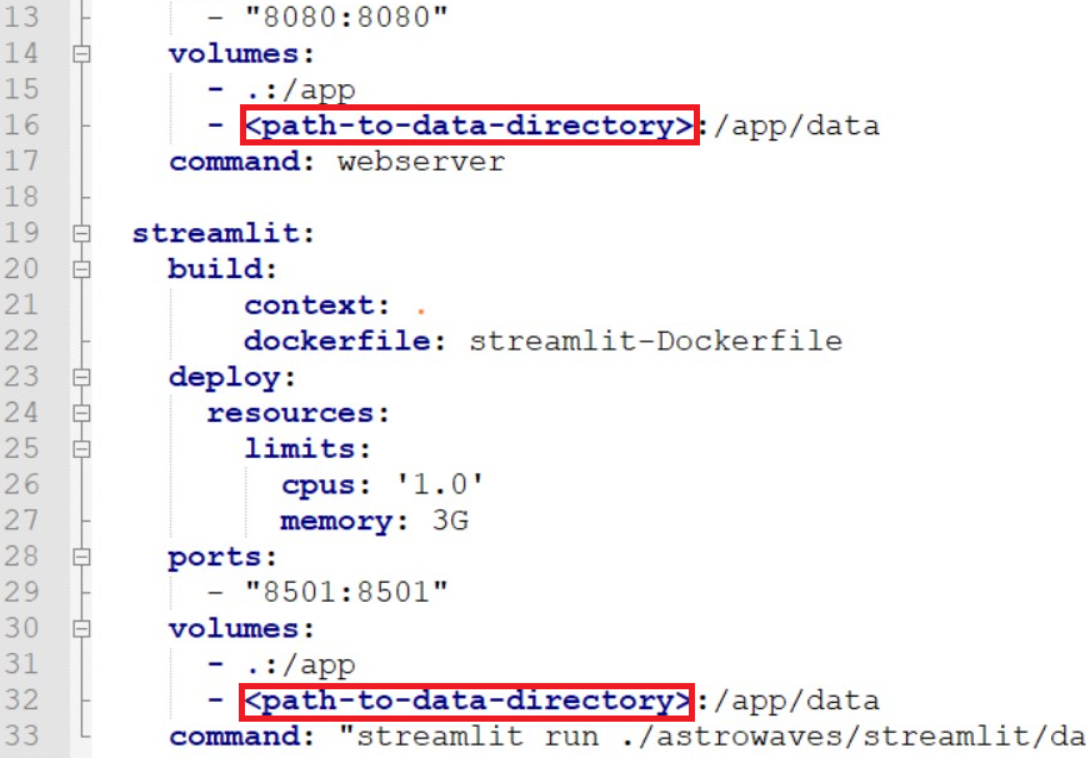
To run the Astral application, it is necessary to perform following steps.

First thing to do is to edit the **docker-compose.yml** file, which is located in the parent directory of the application. The file can be opened using any text editor.

The file specifies the internal configuration of the application. It consists of many values, which should not be modified, but some values are required to be customized.

One of such values is the path to the data directory, in which the image series files will be stored.

To run the Astral application, it is necessary to launch Powershell terminal. Following steps are necessary to launch the program:



Enter the path to the data directory, which contains the files to be processed by Astral. The path needs to be entered in two places (line 16 and line 32). **Do not remove the *:/app/data part* from the lines.**

This step is done once and does not need to be repeated, unless the data directory for image series is changed. In case the data directory is moved to another location, then it is necessary to modify the paths in the docker-compose.yml file.

1. Make sure Docker Desktop application is running, and enough RAM and swap memory is assigned.
2. Inside the Powershell command prompt, navigate to the Astral parent directory. To do this, type:

cd “<path-to-Astral-parent-directory”

For example:

cd “C:\Users\user\Documents \Astral”

If the command was executed successfully, a path specified from the command should be shown next to the cursor.

1. Astral application needs to be built – for that to happen, type:

Docker compose build

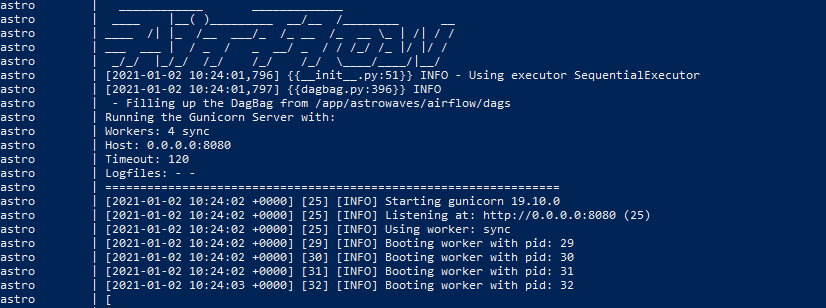


The application building process will start. The process can take up to several minutes, until the command prompt will be available for typing again.

1. Start the application by typing:

Docker compose up

Then, in the Powershell window, the application log files will start to show. The running application will have the similar output at the window:



You can access the application on the URL address: http://localhost:8080/admin/ for Airflow application.

And on the URL address: <http://localhost:8501/> for the Streamlit application.

# Usage

Astral solution consists of two applications, that are accessed under two different URLs in the web browser:

* Airflow – the application for data processing and generation, accessible at <http://localhost:8080/>
* Streamlit – the application for visualization, accessible at <http://localhost:8501/>

WARNING

As a prerequisite of using Streamlit application for a given microscopis timelapse image series, it is necessary to perform a part of processing pipeline in the Airflow application

## Airflow application

### Overview

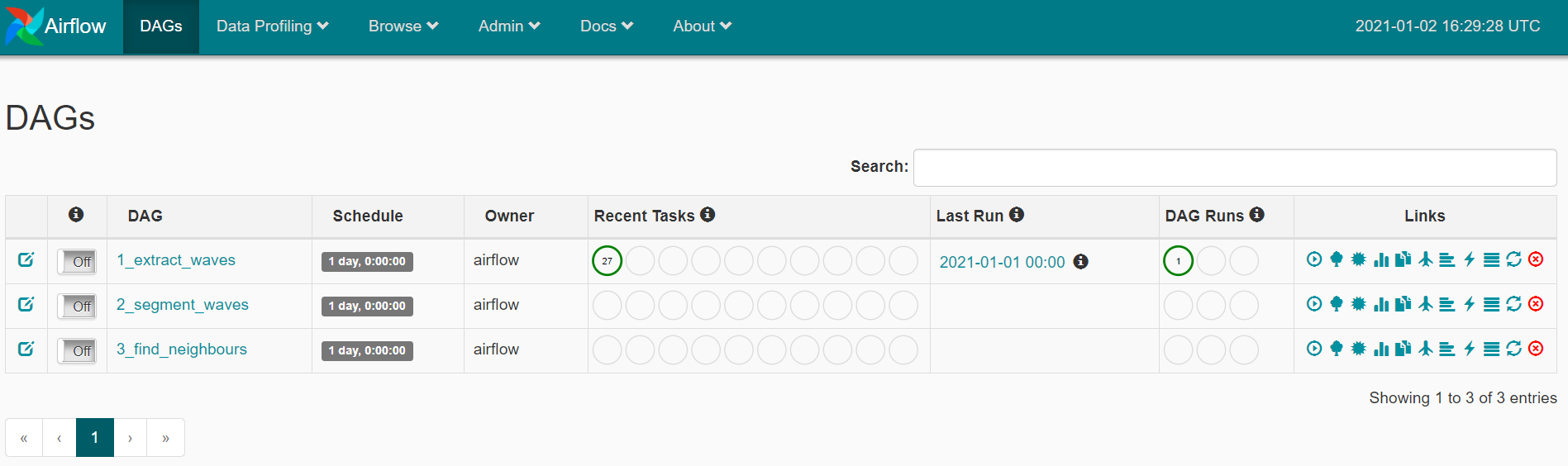
Airflow application will be used as a processing pipeline, which performs subsequent steps to generate an analytic output about the data. Some methods present in the Airflow application will be used to generate intermediate data for the next steps as input, whereas other steps will be responsible for end-data generation. Airflow is designed as a framework for data engineering and is a perfect solution for working with scientific data, as requirements and necessities often change or arise in scientific world.

Airflow provides a toolset for defining and creating computer programs which load, handle, transform and generate data, so microscopic timelapse astrocytic events data is a perfect candidate for developing methods of analysis of such data.

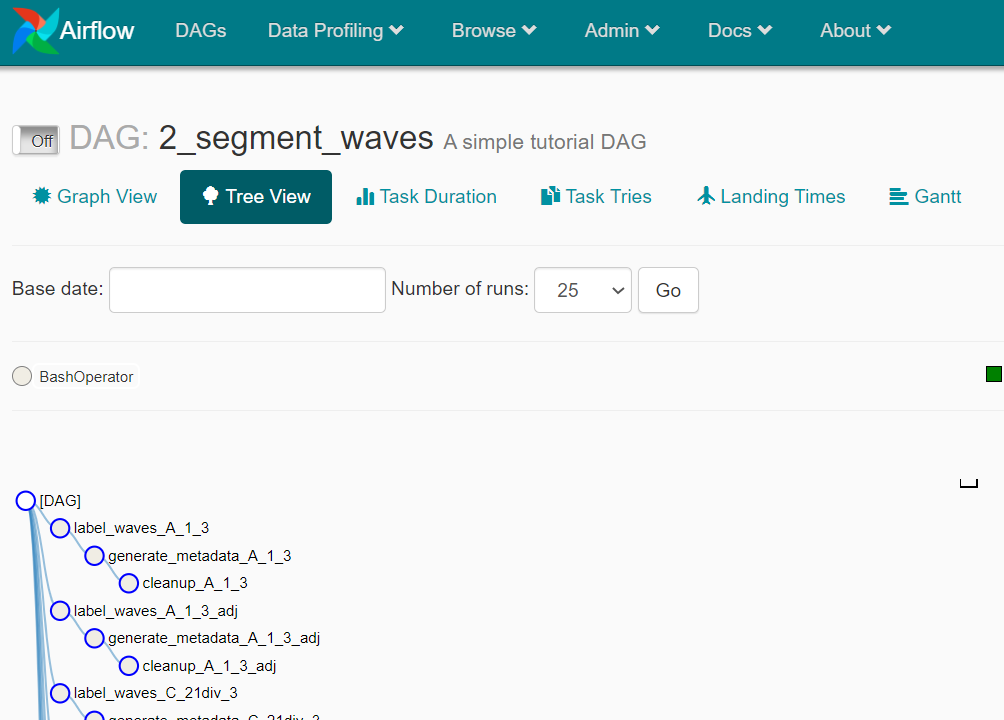
Airflow provides a couple of levels of abstraction. For the time being, a notion of Directed Acyclic Graphs (DAGs) and tasks should be introduced.

DAG defines a complete set of functionalities which accomplish a specific task. A DAG consists of task – computer programs which are responsible for performing a specific task in the DAG’s workflow. DAG may consist of several tasks, which are connected with each other (mostly sequentially) to generate some final outcome.

What’s more, Airflow is a perfect option for a community-driven platform – any contributor may create a new functionality and easily incorporate it into Airflow. This may potentially result in a robust, extensive framework, which can be used for many intents and purposed. Although the initial goal of Astral was to provide a toolset for anaylizing astrocytic events data, it Airflow makes it possible to develop a toolset for any kind of similar data. This makes Astral a very strong tool in future experiments and research in general.



Airflow’s main page displays a list of DAGs available to be executed. By clicking into the DAGs name, the website redirects to the DAG’s workflow and execution details:



The DAG’s details page presents a set of tasks, which are needed to be executed for a DAG to finish.

### Pipeline

The pipeline is divided into 3 parts: “1\_extract\_waves”, “2\_segment\_waves” and “3\_find\_neighbours”. Each DAG consists of several steps, called tasks, which accomplish a specific operation needed to generate a final output. This architecture is clean and easy to be modified, or extended, so it is easy to extend the functionality of the application.

For the time being, the DAGs are defined by the following tasks:

1. 1\_extract\_waves:

* Create\_timelape – this step deals with converting common data formats into scientific Numpy format, which was created by the creators of numpy, a popular Python scientific package. Currently, only Tiff data format is supported, in which consecutive microscopic images as saved as imge series. This step converts such tiff file into a numpy array and saves it for further usage.
* Extract\_waves – this step handles search and extraction of the regions associated with the calcium events. A mean and standard deviation for each pixel is calculated in terms of its value across all the image series. Then, basing on the standard deviation threshold, defined by the user, the pixels are either treated as calcium events, or as a background.

Where i and j denote the positon of the pixel in the image (i represents the x coordinate, whereas j represents y coordinate)

If the pixel has a value greater or equal to the T value, it is considered a calcium wave. In other case, it is considered a background.

All the pixels that are considered as background are set to 0. The pixels treated as calcium waves are preserved. The resulting space is saved as Numpy array, as waves.npy file.

* Create\_masks – the goal of the task is to group together adjacent calcium wave pixels into one single astrocytic event. In 3D space, using a Volume-like algorithm and 3D connectivity, a single value is assigned to each pixel, which denotes its ID. Additionally, if the use\_watershed flag is set, the Watershed algorithm is used in case and two independent calcium waves are connected together. The algorithm’s goal is to detect such cases, and separate the waves.

The task outputs a labelled space, saves as “labelled\_waves.npy” file, and “black\_and\_white.tiff” file, which depicts calcium waves in white, and background in black.

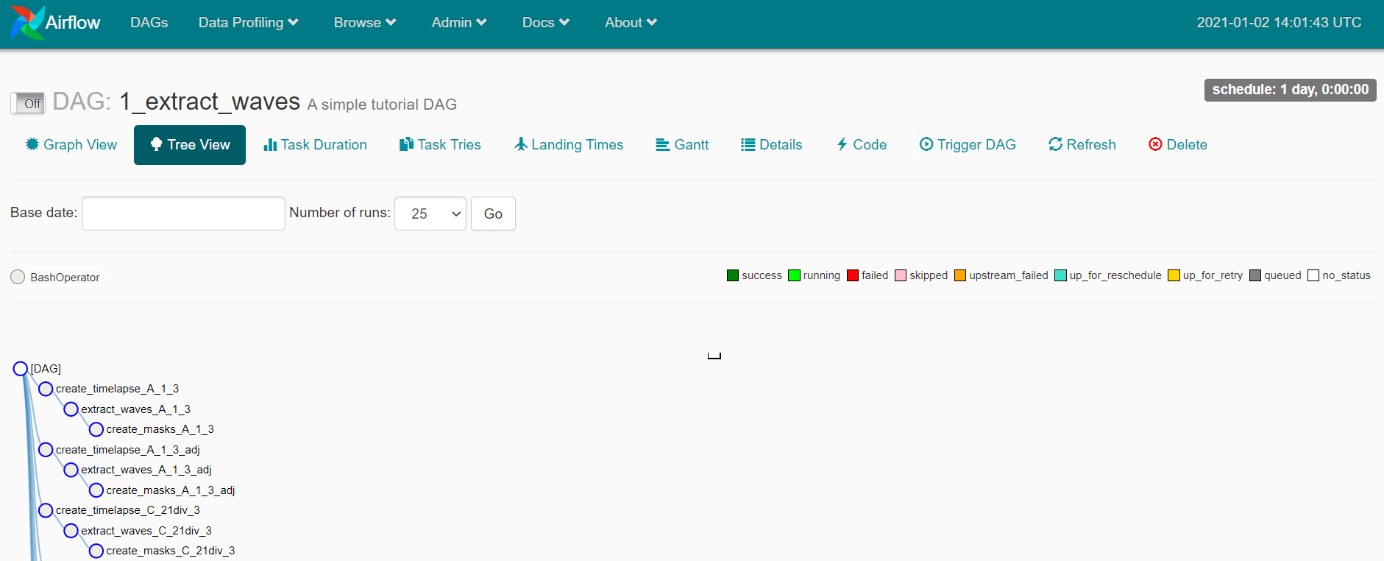
1. 2\_segment\_waves:

* Label\_waves – the goal of the algorithm is to get the coordinates of each pixel of the given calcium wave and save them in the temporary matrix-like object. The resulting file is “waves\_inds.pck” file.
* Generate\_metadata – the goal of the task is to create a tabular dataframe object to store a coordinate of each pixel of the given calcium wave, together with its intensity. Additionally, for each calcium wave, extreme points, together with Euclidean center and center of mass points are calculated for further analysis.
* Cleanup – the goal of the task is to delete any temporary files created in the analysis

1. 3\_find\_neighbours:

* Find\_neighbours – the task searches for the nieghoburs of each calcium wave by creating a bounding box, defined by the tolerance\_xy and tolerance\_z values defined by the user. The center of the bounding box is located at the Euclidean center of the calcium wave. For the extreme points of the wave, a bounding box extends in each direction by tolerance\_xy and tolerance\_z values, accordingly. Any calcium wave located inside defined volume is treated as a neighbours of that calcium wave. The task produces neighbours.csv file, which lists all the neighbours for the given calcium wave, together with basic information, such as distance between centers of the calcium waves, in xy and t axes respectively. An additional file, named “neighbour\_statistics.csv” is also generated, with a summary for each calcium wave.
* Find\_repeats – The task is an extension of the previous step, which works sequentially – for each group of neighbours, it checks the intersection between a shape and its neighbours. The calcium waves are projected into an xy plane beforehand. If the Intersection Over Union for the two shapes is greater than Intersect\_threshold parameter specified by the user, the calcium waves are considered repeats of each other. As a result, the task assigns a “single” category for single calcium waves, and “repeat” category for repeated calcium waves. Temporary “repeats.pck” and “singles.pck” files are generated.
* Generate\_csvs – The task produces final two csv files for repeated calcium waves and single calcium waves. Some morphological features are calculated for each object (for repeats, those features are averaged across all repeated instances).

### Usage

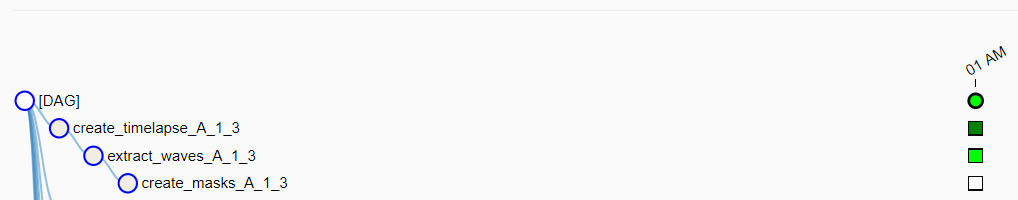


For each file, a set of tasks described in \_\_\_\_ is created. The tasks are executed sequentially. It is also possible to visualize task in a Graph-like manner by clicking “Graph View” tab.

To run a DAG, it is necessary to first launch it by changing its state using a ON/OFF switch in the top left position in the image:



Once clicked, the switch changes its state to On, and the DAG will start executing the tasks.



Green circle at the top-right corner indicates that DAG is running and there are remaining tasks to be executed. Dark green box next to the “create\_timelapse\_A\_1\_3” task indicates, that the task of creating timelapse for the file “A\_1\_3” has ended with success. Light green box next to the “extract\_waves\_A\_1\_3” indicates, that the task is currently running. Blank boxes indicate that the task has not started yet.

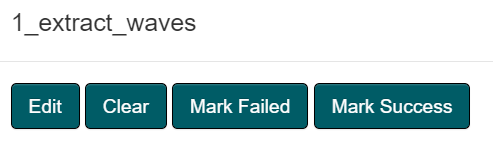
The full legend of colors is listed on the top of the DAG scheme:



It is possible to stop a running DAG by clicking into the instance circle at the top of the task’s status list:



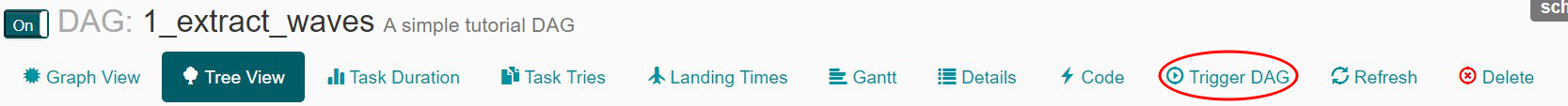
Once clicked, a menu will show with an option of marking a DAG as failed or completed successfully:



Both options will result in terminating a DAG.

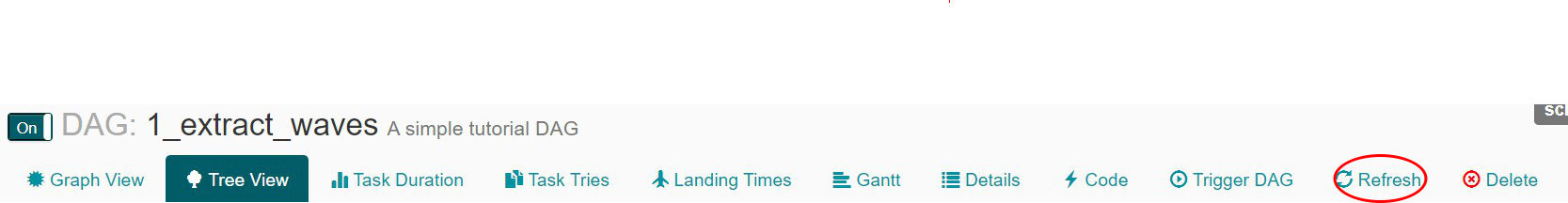
Important

In case DAG needs to be executed again, a Trigger DAG button should be clicked. Switching off and on will not invoke DAG again, however, to trigger a DAG, it is necessary to leave a DAG on the ON mode.



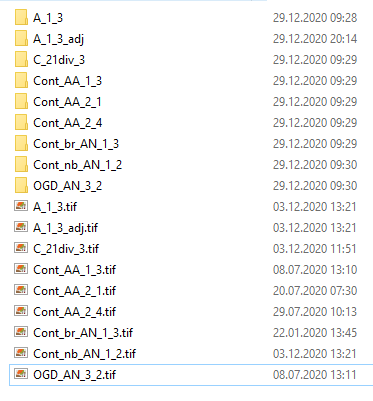


If during execution new files were added and are not visible in the DAG’s task tree, it is possible to refresh the DAG by using the Refresh button:

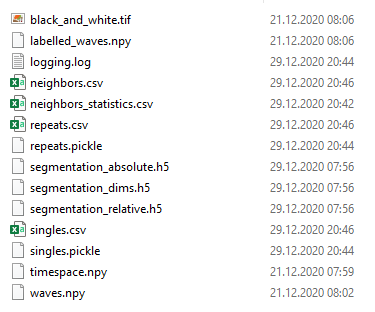


After refreshing, all the new files should be visible in the DAG’s tree.

For each file processed in the Airflow application, a directory in a data folder specified in docker-compose.yml is created automatically:



All the files generated in the processing pipeline will be saved in these folders.



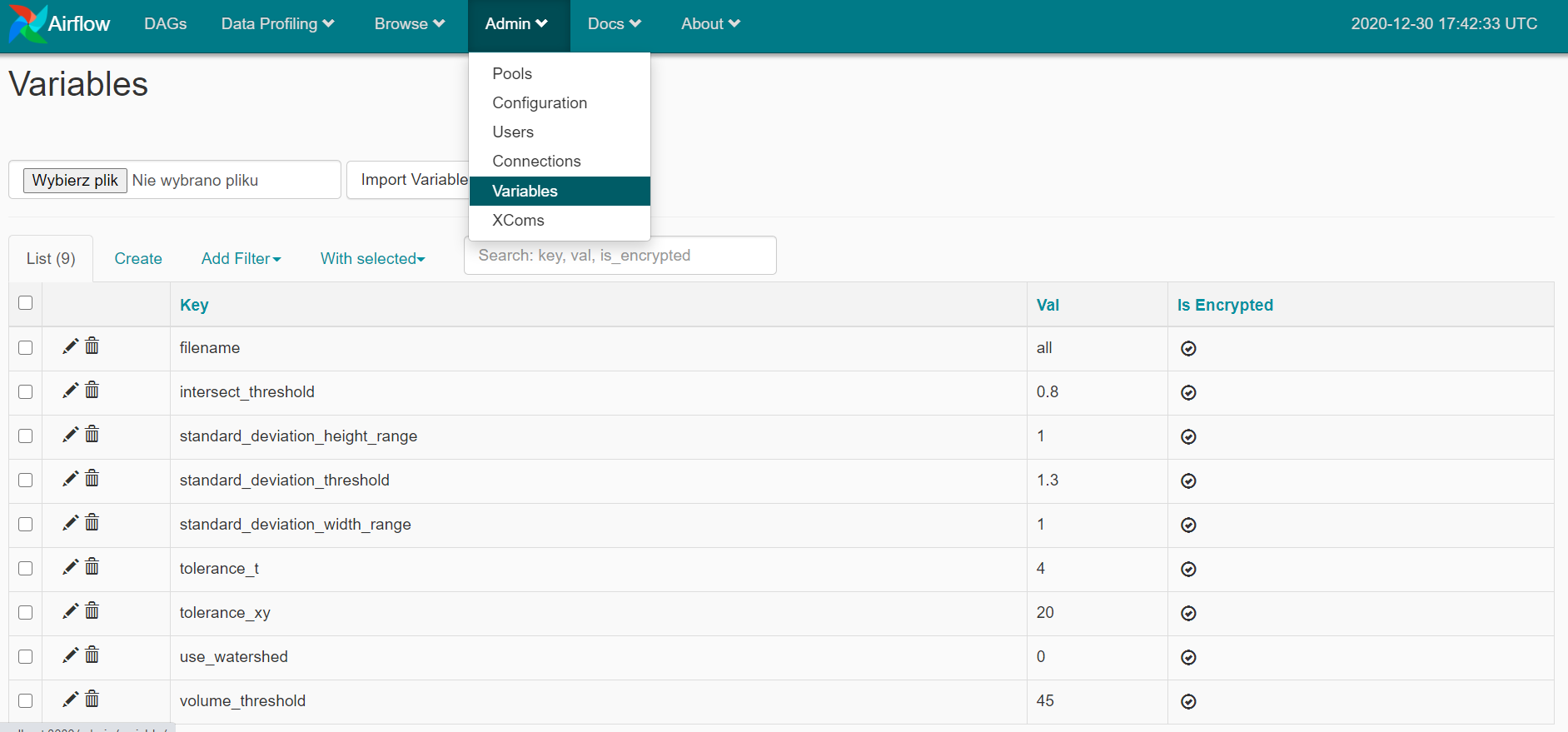
Each file and its application has been described in \_\_\_ chapter. The files are required for the processing pipeline as well as for Streamlit application to run properly.

### Airflow configuration

Airflow provides a built-in functionality of configuring the pipeline for a specific user. A table with short description of each customizable parameter is provided below. A full description of the parameters is listed later in the manual.

|  |  |  |
| --- | --- | --- |
| Parameter Name | Description | Default Value |
| [Filename](#_Filename) | A filename of a timelapse sequence to be processed from the data directory. Leave ‘all’ if all the files should be processed | All |
| [Intersect\_threshold](#_Intersect_threshold) | How much should the calcium waves overlap in their z-projection to be treated as repeats. | 0.8 |
| [Standard\_deviation\_height\_range](#_Standard_deviation_height_range) | How many adjacent pixels in height dimension to take into account during std calculation | 1 |
| [Standard\_deviation\_threshold](#_Standard_deviation_threshold) | Standard deviation threshold for detecting calcium events | 1.3 |
| [Standard\_deviation\_width\_range](#_Standard_deviation_width_range) | How many adjacent pixels in width dimension to take into account during std calculation | 1 |
| [Tolerance\_t](#_Tolerance_t) | A maximum distance between subsequent calcium waves to treat them as neighbours | 4 |
| [Tolerance\_xy](#_Tolerance_xy) | A maximum distance in xy plane to treat two different calcium waves as neighbours | 20 |
| [Use\_watershed](#_Use_watershed) | Whether during mask generation, a Watershed algorithm for splitting adjacent calcium waves should be used | 1 |
| [Volume\_threshold](#_Volume_threshold) | Volume threshold in total number of pixels to treat a shape as a calcium wave. Calcium waves below volume\_threshold will be discarded from the analysis. | 45 |

Those parameters can be set in the Variables section in the Admin tab:



To change the parameter value, you may use a Pen icon next to the parameter name.

Important

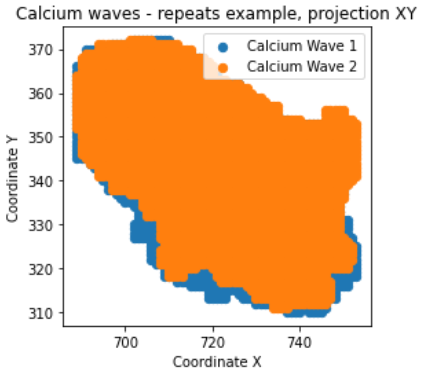
It is possible to override the default parameter values. These values are loaded from the **variables.json** file, located in the parent directory of the application. Modifying the values inside the **variables.json** file will change the default values of the application from the next startup.

#### Filename

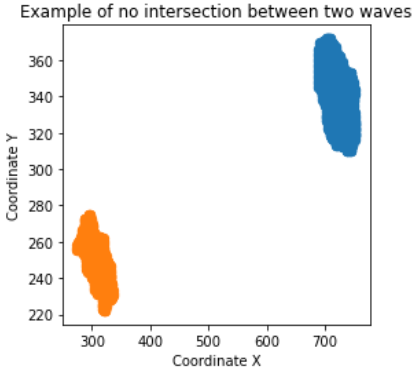
This parameter specifies the filename of the image series to be processed. If all the images inside the data directory should be analyzed, a special value for the parameter has been defined. To process all the files, “all” value should be entered.

#### Intersect\_threshold

Defines, what is the threshold for for the Intersection Over Union between projections of the two calcium waves to consider those waves as repeats.



In the aboce example, the projections of the two calcium waves heavily overlap with each other. The value of intersection over union is high in this case, and if is greater than Intersect\_threshold, those will be treated as repeats.



In the above case, the Intersection Over Union is equal to 0, as there is no overlap between the waves. Unless the Intersect\_threshold is set to 0, those waves will not be treated as repeats.

#### Standard\_deviation\_height\_range

This parameter specifies, how many consecutive pixels in the image should be taken into account during thresholding in y axis. The thresholding is the process of extracting signalling events from the image series. The parameter is used in ‘extract\_waves” task in “1\_extract\_waves” DAG.

#### Standard\_deviation\_threshold

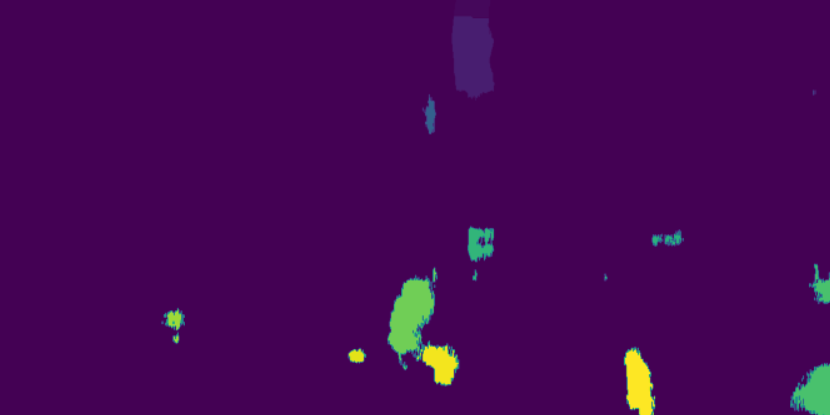
The threshold value for the standard deviation, above which a pixel is treated as calcium event. Pixels with values below the standard deviation threshold will be treated as background.

#### Standard\_deviation\_width\_range

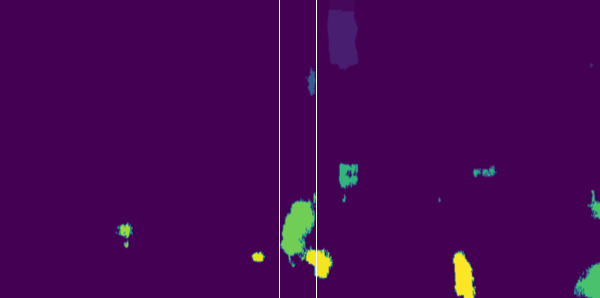
This parameter specifies, how many consecutive pixels in the image should be taken into account during thresholding in x axis. The thresholding is the process of extracting signalling events from the image series. The parameter is used in ‘extract\_waves” task in “1\_extract\_waves” DAG.

#### Tolerance\_t

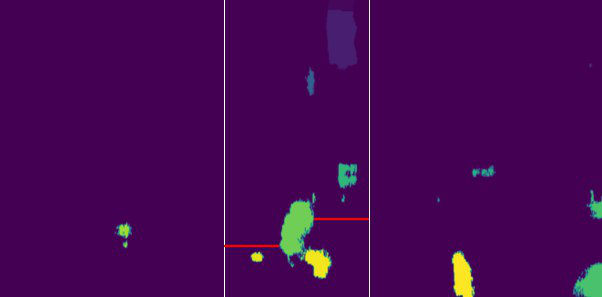
This parameter is used in “find\_neighbours” task in “3\_find\_neighbours” DAG. This parameters specifies, how far can calcium waves be separated in terms of t coordinate to be considered neighbours. For example:



For a green shape, a bounding box with respect to the t coordinate is drawn:



A bounding box is enlarged by tolerance\_z parameters from both sides:



The red lines’ length is specified by tolerance\_z parameter. Any calcium wave that is contained inside the defined region, is considered a neighbour.

#### Tolerance\_xy

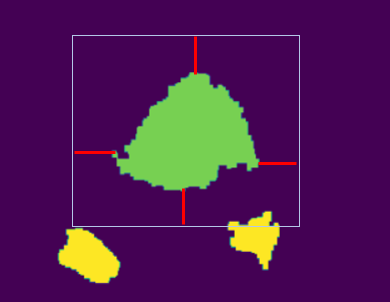
This parameter is used in “find\_neighbours” task in “3\_find\_neighbours” DAG. This parameter specifies, how far can calcium waves be separated in terms of x and y coordinate to be considered as neighbours. For example:



In the image, Astral has detected multiple calcium waves, marked with distinguishing colors. For a given calcium wave, a bounding box is drawn:



This bounding box is expanded in both x and y directions – the tolerance\_xy determines how far should the box be expanded:



The red lines indicate the size of the enlargement of the bounding box – each of those lines expanded the bounding box by tolerance\_xy parameter. In other words, the tolerance\_paramer has enlarged the bounding box by double the tolerance\_xy value in both x and y coordinates. Any shape that either crosses the region defined by the bounding box is considered a neighbour. In the case above, one of the waves is a neighbour, as it crosses the boundaries, whereas the other one is not considered a neighbour.

#### Use\_watershed

This parameter is used in “Create\_masks” task in “1\_extract\_waves” DAG. If the parameter is set to 1, the task additionally uses Watershed algorithm for splitting adjacent waves apart. If the parameter is set to 0, the task does not perform this additional step. Below you may find an example of a 2D scenario for a Watershed algorithm:

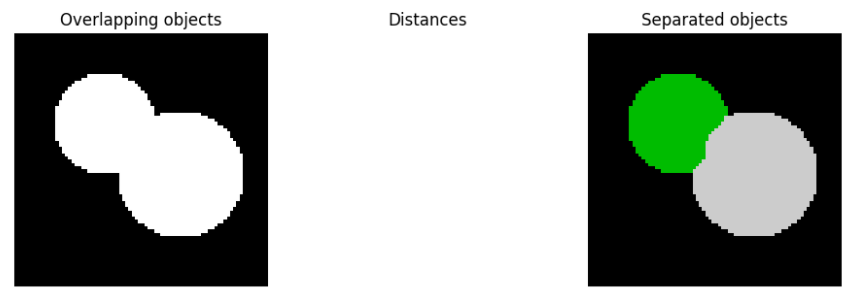


Figure 1. Splitting of overlapping objects. Source: <https://scikit-image.org/docs/dev/auto_examples/segmentation/plot_watershed.html>

The algorithm works in 3D scenario. Basing on intensity of the pixels forming a calcium waves, it determines whether a case contains multiple waves joined together.

#### Volume\_threshold

Volume threshold is used in “label\_waves” task in “2\_segment\_waves” DAG. It drops calcium waves, which consist of less pixels than volume\_threshold. If the sum of all wave’s pixels (across all timesteps) is less than the threshold, it is disregarded from further analysis.

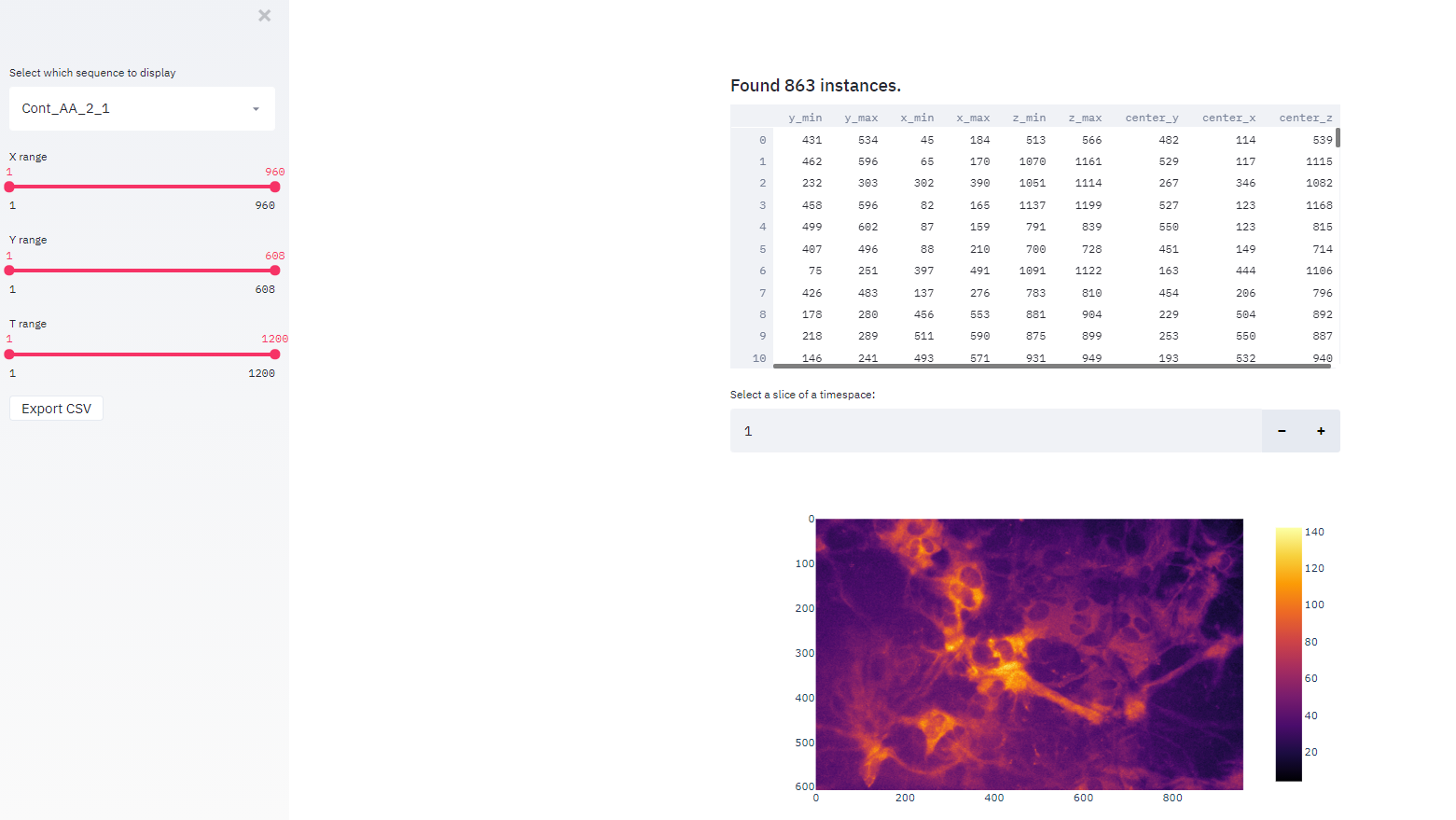
## Streamlit application

### Overview

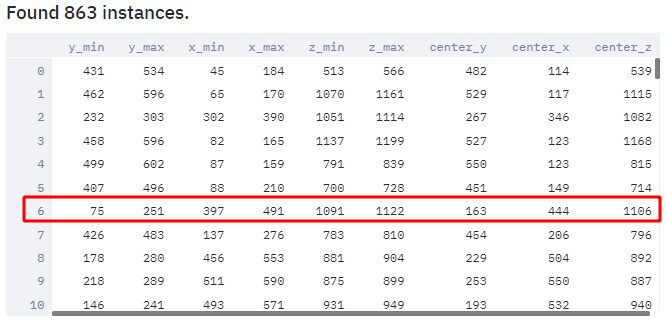
The purpose of Streamlit is to deliver a simple, yet powerful solution for visualizion and insight into the processed data. Streamlit provides a useful navigation through the astrocytic timelapse, simplifies the process of localization of particular astrocytic events by tabular visualizations, local filtering and markdown. Furthermore, Streamlit provides a 3D visualization of astrocytic events – a particularly useful feature for dynamics study and configuration troubleshooting.

This simple, yet powerful tool fills a gap in the Airflow processing pipeline, which is a lack of immediate visual navigation through the image series. Using Airflow-generated data, Streamlit provides a link between tabular data and imagination. As tabular data is important for problem statement and experiment interpretation, it does not provide a visual representation for the generated data. Therefore, the intention of Streamlit was to compensate the shortcomings of the processing pipelines, such as Airflow.

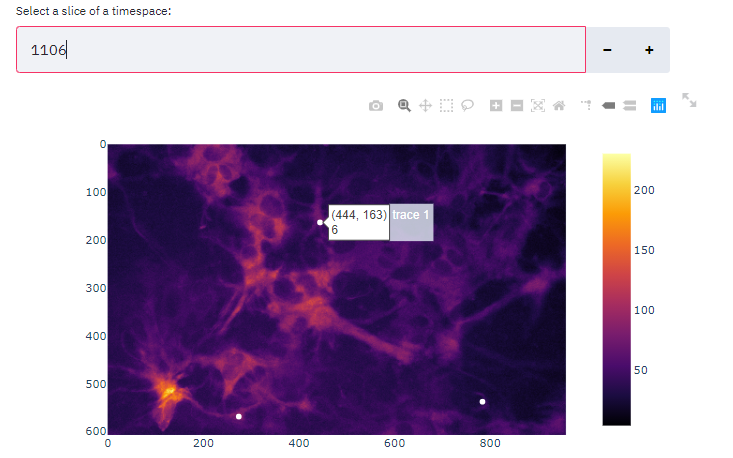
### Usage



Streamlit application enables the user to inspect the timelapse in terms of detected calcium waves. First, a list of all the calcium waves with their ids and coordinates is shown. To check the location of a given calcium wave in the timelapse, it is possible to navigate through the timelapse. For example:

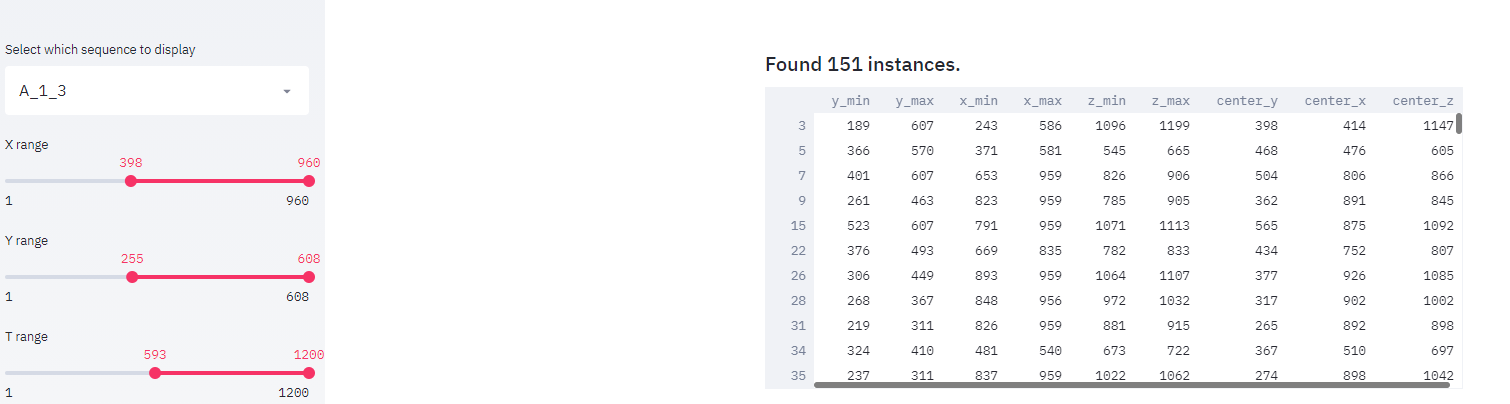


We would like to find the location in the series of the calcium wave with the id of 6. To do that:



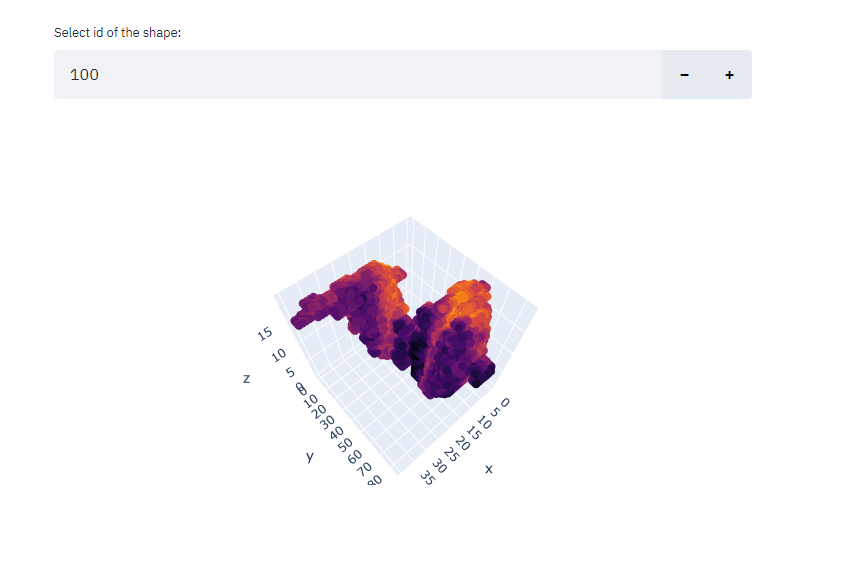
It is necessary to select a slice of a timespace of the center t coordinate of that calcium wave. Then, the calcium wave is marked with a white dot. In this case, two more calcium waves are also present on the slice.

Three sliders on the left can be used to filter the calcium wave list.



In this case, ranges in all the dimensions have been reduced. Only those calcium waves that are contained in this constrained subspace, are listed.

It is possible to view any calcium wave in 3D by selecting in by its id:



In this case, a calcium wave of id 100 is shown.

IMPORTANT NOTE

Streamlit application uses files generated in the “1\_extract\_waves” DAG and “2\_segment\_waves” DAG. It is not required to run “3\_find\_neghbours” DAG for the visualization in Streamlit application.