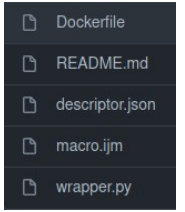
Adding a workflow to BIAFLOWS

Defragmentation:

bringing BioImage Analysts to the cloud!

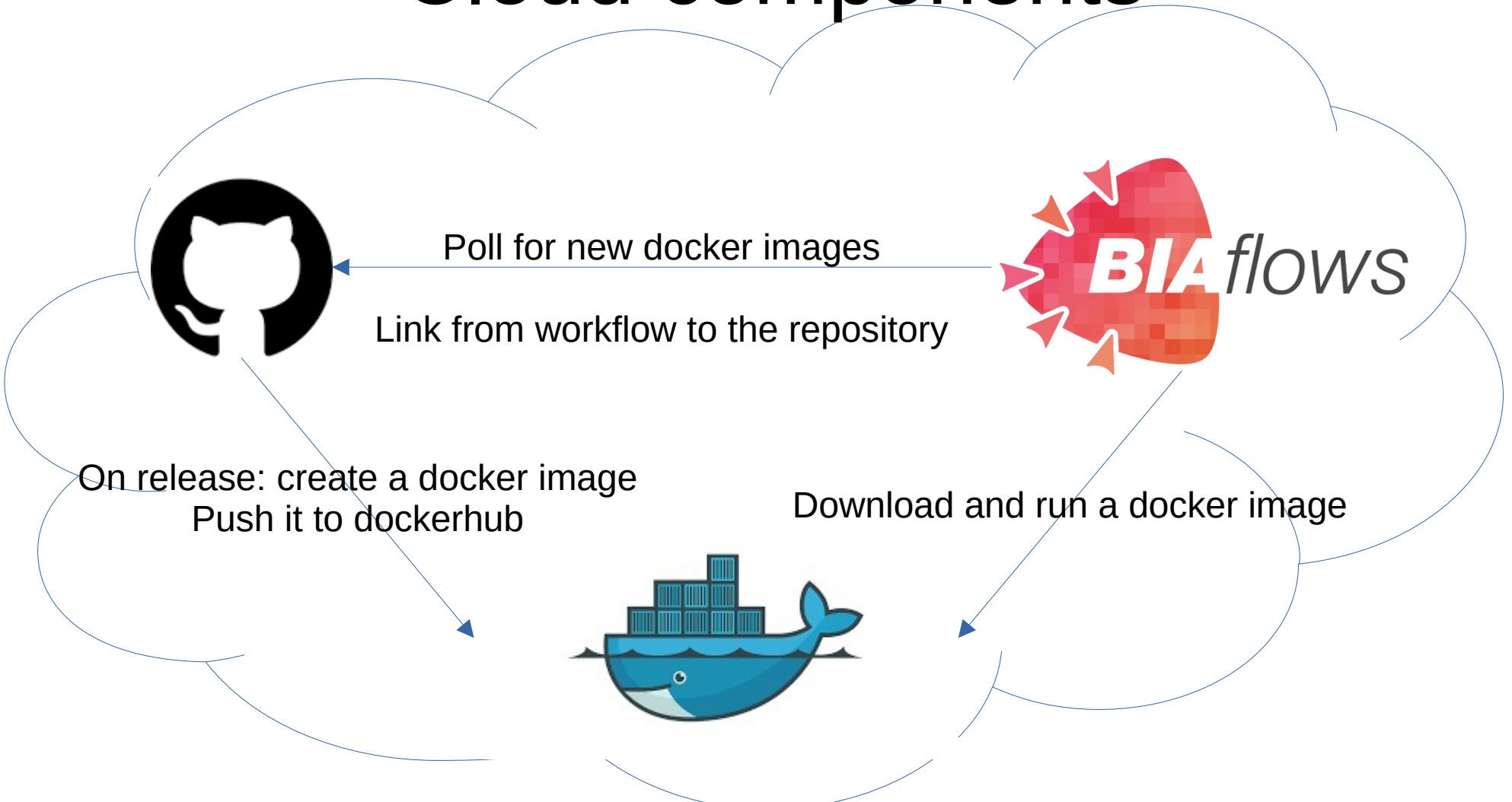
Overview



- Prerequisites:
  - A github account
  - A dockhub account
  - configure BIAFLOWS to scan your github repositories
- Create a new repository on github
  - Give the new repository the right to access your dockerhub
- Modify your workflow to
  - Run on the images in a folder
  - Accept input-parameters
  - Produce results in the expected form

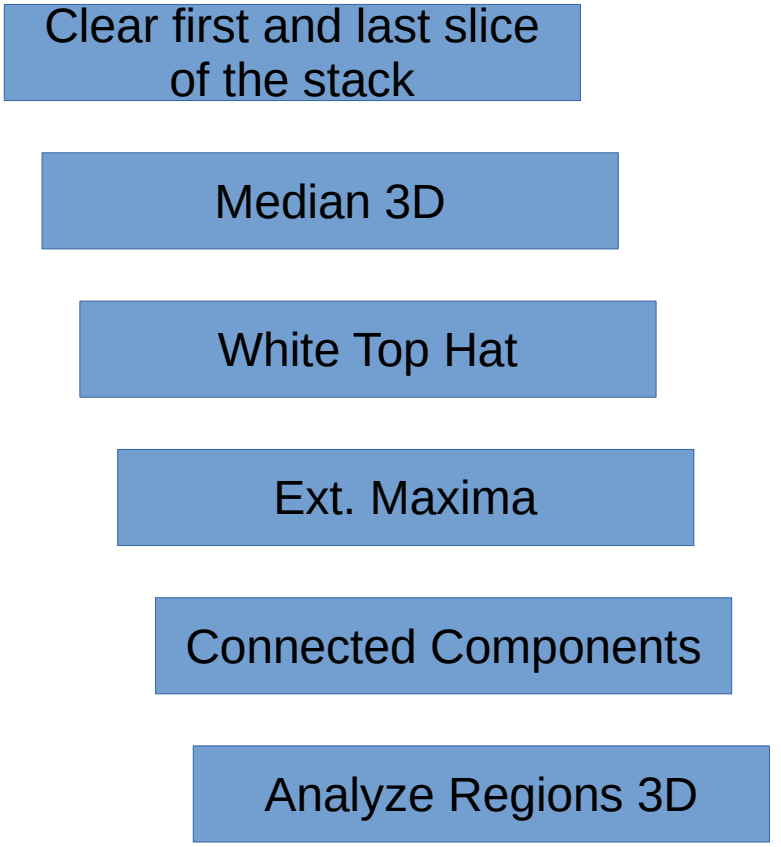
- Add 4 files to the github repository
  - Dockerfile
    - Defines the computational environment
  - wrapper.py
    - Download images, run workflow, create/upload annotations, create/upload metrics
  - macro.ijm
    - the image analysis workflow
  - descriptor.json
    - Definition of the parameters of the workflow
- Make a release
  - A release triggers a git-hub action that
    - creates a docker-image from the repository
    - Pushes the docker-image to dockerhub

Cloud components



Example
ImageJ – 3D Spot Detection

- Basic idea
  - Apply Top-Hat-Transform
  - Find maxima
- Parameters:
  - Radius median in xy
  - Radius median in z
  - Radius top-hat in xy
  - Radius top-gat in z
  - Dynamic (ext. maxima)
  - 3D connectivity 6 or 26



Example – The original macro

```
noise_filter_radius_xy = 1;
noise_filter_radius_z = 0.5;
top_hat_radius_xy = 7.5;
top_hat_radius_z = 3.5;
dynamic = 75;
connectivity = 6;


getDimensions(width, height, channels, slices, frames);
Stack.setSlice(1);
run("Select All");
run("Clear", "slice");
Stack.setSlice(slices);
run("Clear", "slice");
run("Select None");
run("Median 3D...", "x="+noise_filter_radius_xy+" y="+noise_filter_radius_xy+" z="+noise_filter_radius_z);
run("Morphological Filters (3D)", "operation=[White Top Hat] element=Cube x-radius="+top_hat_radius_xy+" y-radius="+top_hat_radius_xy+" z-radius="+top_hat_radius_z);
run("Extended Min & Max 3D", "operation=[Extended Maxima] dynamic="+dynamic+" connectivity="+connectivity);
run("Connected Components Labeling", "connectivity="+connectivity+" type=[16 bits]");
run("Analyze Regions 3D", "centroid surface_area_method=[Crofton (13 dirs.)] euler_connectivity=6");
```

dockerhub

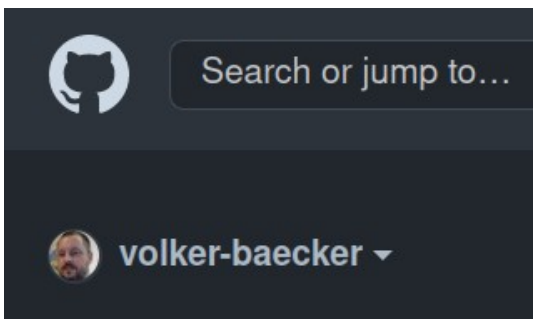
- Docker hub hosts and serves built docker-images
- You need a dockerhub account with the same name as your github account
  - dockerhub allows only letters and digits
  - ‘-’ on github will be removed for dockerhub
  - everything will be made lower-case for dockerhub

  volker-baecker volkerbaecker

- Organizations can also be used on both sites

  Neubias-WG5 neubiaswg5

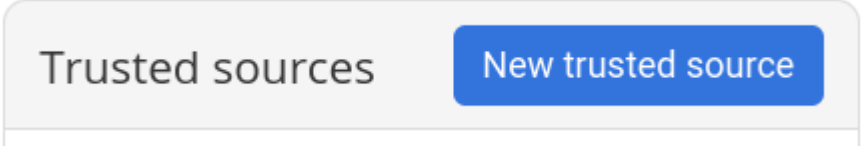
- Create a dockerhub account that corresponds to your github account!



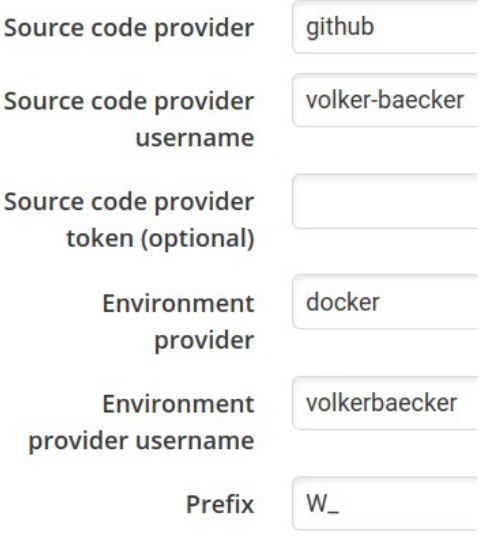
Configure BIAFLOWS to scan your repositories

- In BIAFLOWS
  - Go to Workflows

  - Add a new trusted source



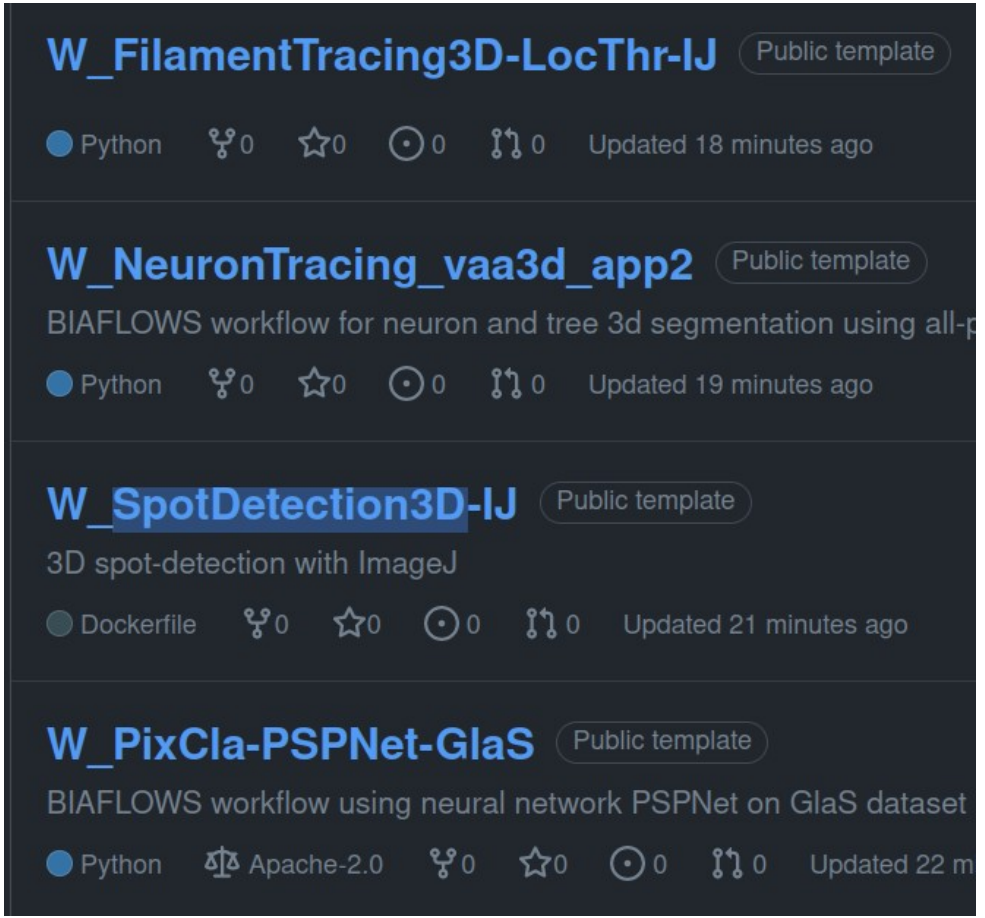
- Fill in
  - Your github user-name
  - Your dockerhub user-name
  - The prefix of workflows to import

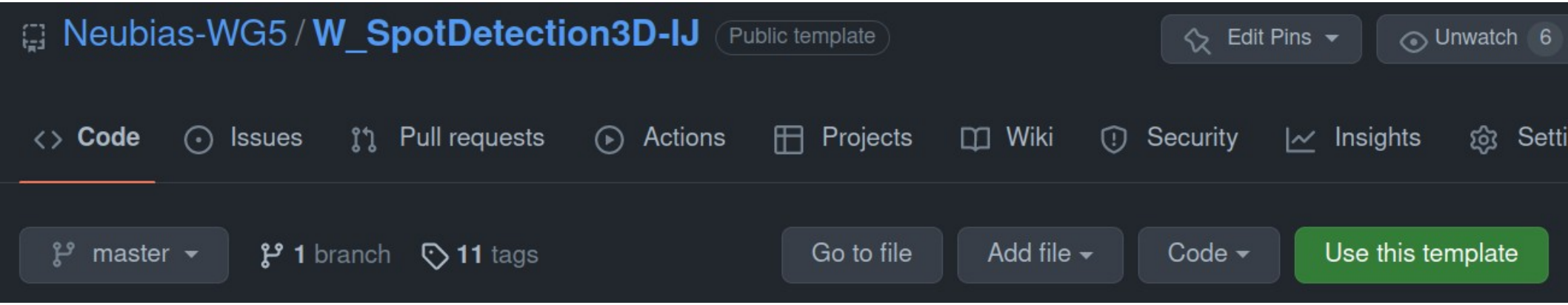


Create a new repository on github

- Look for the most similar NEUBIAS-WG5 workflow repository

https://github.com/orgs/Neubias-WG5/repositories





- Press the “Use this template” button

- Add a name for the new repository, starting with W_
- Modify the Readme

Create a new repository from W_SpotDetection3D-IJ

The new repository will start with the same files and folders as Neubias-WG5/W_SpotDetection3D-IJ.

Owner * **Repository name** *

volker-baecker / W_SpotDetection3D-IJ-TopHat

Great repository names are short and memorable. Need inspiration? How about **expert-palm-tree**?

Description (optional)

Public
Anyone on the internet can see this repository. You choose who can commit.

Private
You choose who can see and commit to this repository.

Include all branches
Copy all branches from Neubias-WG5/W_SpotDetection3D-IJ and not just `master`.

You are creating a public repository in your personal account.

Create repository from template

.github/workflows	Initial commit	4 minutes ago
Dockerfile	Initial commit	4 minutes ago
IJSpotDetection3D.ijm	Initial commit	4 minutes ago
README.MD	Update README.MD	3 minutes ago
descriptor.json	Initial commit	4 minutes ago
wrapper.py	Initial commit	4 minutes ago

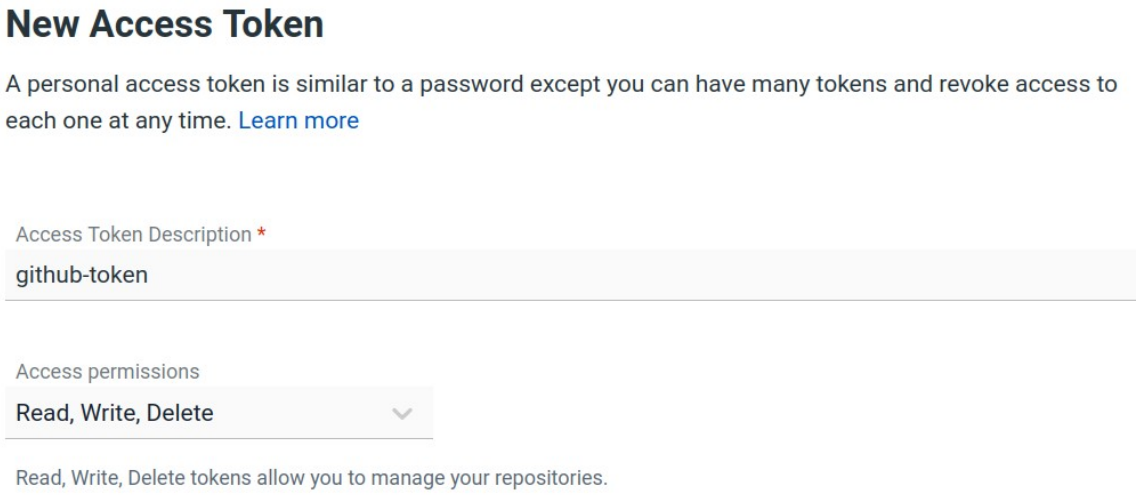
README.MD

W_SpotDetection3D-IJ-TopHat

This workflow detects spots in a 3D image by filtering the image by a Top-Hat-Filter (user defined radius) and detecting local intensity maxima with a given dynamic (user defined threshold).

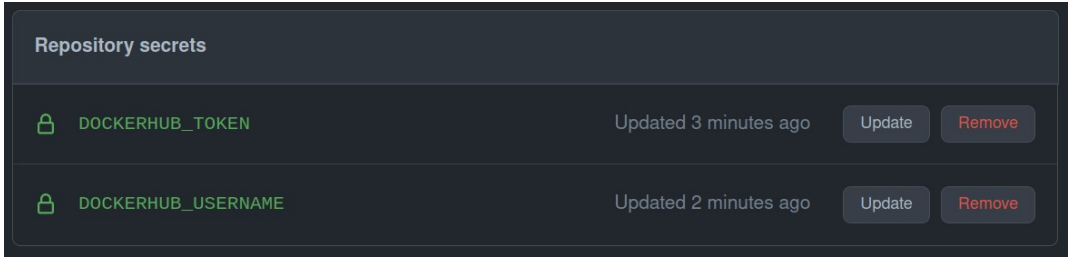
Give the new repository the right to access your dockerhub

- Log in to dockerhub
- Click on your username
  - In the upper right corner
  - Go to “Account Settings”
- Select Securtity
- Press “New Access Token”
- Enter a name for the token
- Generate the token
- **Copy the token**



Give the new repository the right to access your dockerhub

- Go back to your github-repository
- In “Settings”
  - Go to “Secrets>Actions”



- Create a secret with the name DOCKERHUB_TOKEN and paste the copied token as its value
- Create a secret with the name DOCKERHUB_USERNAME and your dockerhub username as value
- Instead of a doing it for each repository you can do it for an organization

Adapt your workflow

- Replace the macro in the git-repository with your macro
- Modify it to produce the expected result
  - Mask of centroids
- Modify the macro to accept parameters from the command-line
- Modify it to work on all images in the input folder and write the results to the output folder

Create the expected result

- What is the expected result?
- You find the information in the BIAFLOWS doculentation, see:

https://neubias-wg5.github.io/problem_class_ground_truth.html#steps-section

- In our case
  - A 16-bit image
    - With a pixel of 65535 for each centroid of a spot
    - 0 for the background

Create the expected result

- Our macro just measures the centroids of the spots and reports them in a table
- From the table we create the output image, by setting a pixel to 65535 for each centroid in the table

```
columnNames = split(Table.headings, "\t");
X = Table.getColumn(columnNames[1]);
Y = Table.getColumn(columnNames[2]);
Z = Table.getColumn(columnNames[3]);
Table.sort(columnNames[3]);
run("Select All");
run("Clear", "stack");
run("Select None");
for (c = 0; c < X.length; c++) {
    x = X[c];
    y = Y[c];
    z = Z[c];
    Stack.setSlice(z);
    setPixel(x, y, 65535);
}
```

Read parameters from the command-line

- Our macro has the parameters
  - noise_filter_radius_xy
  - noise_filter_radius_z
  - top_hat_radius_xy
  - top_hat_radius_z
  - dynamic
  - Connectivity
- In addition we need parameters for the input and output folders
  - inputDir
  - outputDir

```
// Path to input image and output mask
inputDir = "/media/baecker/DONNEES/mri/2022/neubias/data";
outputDir = "/media/baecker/DONNEES/mri/2022/neubias/out";

// Functional parameters
noise_filter_radius_xy = 1;
noise_filter_radius_z = 0.5;
top_hat_radius_xy = 7.5;
top_hat_radius_z = 3.5;
dynamic = 75;
connectivity = 6;

arg = getArgument();
parts = split(arg, ",");

for(i=0; i<parts.length; i++) {
   nameAndValue = split(parts[i], "=");
   if (indexOf(nameAndValue[0], "input") > -1) inputDir = nameAndValue[1];
   if (indexOf(nameAndValue[0], "output") > -1) outputDir = nameAndValue[1];
   if (indexOf(nameAndValue[0], "filter_xy") > -1) noise_filter_radius_xy = nameAndValue[1];
   if (indexOf(nameAndValue[0], "filter_z") > -1) noise_filter_radius_z = nameAndValue[1];
   if (indexOf(nameAndValue[0], "top_hat_xy") > -1) top_hat_radius_xy = nameAndValue[1];
   if (indexOf(nameAndValue[0], "top_hat_z") > -1) top_hat_radius_z = nameAndValue[1];
   if (indexOf(nameAndValue[0], "dynamic") > -1) dynamic = nameAndValue[1];
   if (indexOf(nameAndValue[0], "connectivity") > -1) connectivity  = nameAndValue[1];
}
```

Run on all images in a folder

- We got the input and output folders as parameters
- Apply workflow to each image in the input folder and write the result to the output folder
- Cleanup after each iteration
- Run in batch-mode
- Quit the java-virtual machine process at the end

```
setBatchMode(true);

images = getFileList(inputDir);



for(i=0; i<images.length; i++) {

   image = images[i];

   if (!endsWith(image, ".tif")) continue;
   open(inputDir + "/" + image);

   ...

   // Export results

   save(outputDir + "/" + image);



   // Cleanup

   run("Close All");

   close(title + "-morpho");

}
run("Quit");
```

Setting up the environment

- We have to install the ImageJ-plugin MorphoLibJ
- Use a more recent version of FIJI

We change the Dockerfile accordingly

```
# Install Fiji plugins
RUN cd /fiji/plugins && wget -O  MorphoLibJ_-1.5.1.jar
https://github.com/ijpb/MorphoLibJ/releases/download/v1.5.1/MorphoLibJ_-1.5.1.jar

# Install Fiji.
RUN wget --no-check-certificate https://downloads.imagej.net/fiji/archive/20221013-0917/fiji-linux64.zip
RUN unzip fiji-linux64.zip
...
# Clean up
RUN rm fiji-linux64.zip
```

Tell the wrapper how to run your workflow

- The wrapper downloads images, runs your workflow and uploads results
- The only thing we need to change is the passing of the parameters from BIAFLOWS GUI to your workflow

```
command = "/usr/bin/xvfb-run java -Xmx6000m -cp /fiji/jars/ij.jar ij.ImageJ --headless --console " \
          "-macro macro.ijm \"input={}, output={}, filter_xy={}, filter_z={}, top_hat_xy={}, top_hat_z={}, dynamic={}, connectivity={}\"" \
          .format(in_path,  out_path,  nj.parameters.filter_xy, nj.parameters.filter_z, nj.parameters.top_hat_xy, nj.parameters.top_hat_z, nj.parameters.dynamic, nj.parameters.connectivity)
```

Adapt the json descriptor

- Fields to be changed
  - Name
  - Image (on dockerhub)
  - Description
  - Command-line (parameters)
  - Inputs (parameters)
    - after cytomine_id_software
    - Set GUI text and default values for the parameters

"name": "SpotDetection3D-IJ-TopHat",

"image": "volkerbaecker/w_spotdetection3d-ij-tophat",

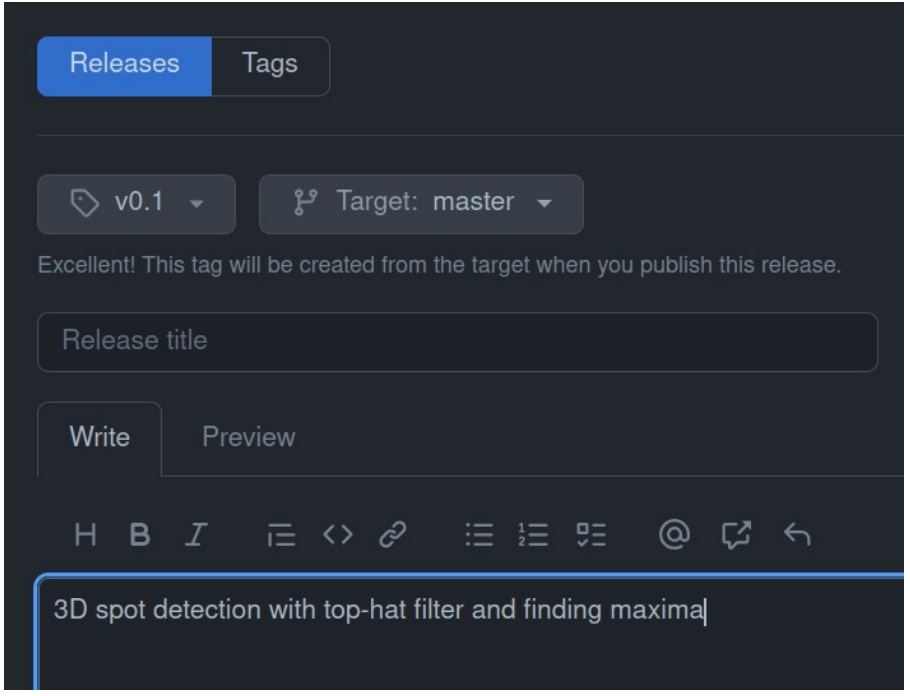
"description": "3D spot detection using a top hat filter and the detection of maxima.",

"command-line": "python wrapper.py CYTOMINE_HOST CYTOMINE_PUBLIC_KEY CYTOMINE_PRIVATE_KEY CYTOMINE_ID_PROJECT CYTOMINE_ID_SOFTWARE FILTER_XY FILTER_Z TOP_HAT_XY TOP_HAT_Z, DYNAMIC, CONNECTIVITY",

```
{
        "id": "filter_xy",
        "value-key": "@ID",
        "command-line-flag": "--@id",
        "name": "Radius xy",
        "description": "Radius for the Median-filter in the x-y-plane",
        "type": "Number",
        "default-value": 1,
        "optional": true
    }, ….
```

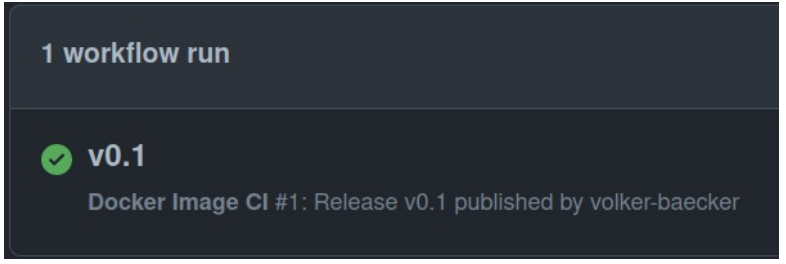
Create a release on github

- Click on tags
- Create a new release
- Use a version number of the form v<x>.<y>.<z>



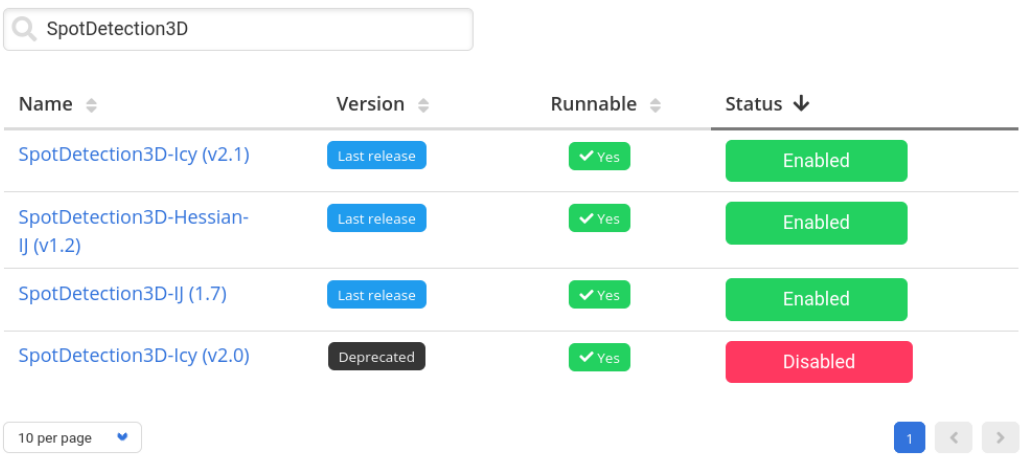
Check if the docker image is built correctly

- In github click on “Actions”
- Look for the version of the release you just created
  - Note: Building the image can take some time



Add the workflow to a problem class in BIAFLOWS

- Login to BIAFLOWS
- Select a problem
  - Problem: SPOT-COUNTING-3D
- Click on configuration
- Click on workflows
- Search your workflow and enable it



Finish

- You can now use your workflow in BIAFLOWS
  - See how it performs (metrics)
  - Compare the results with other workflows
- Run the workflow locally on your machine, using docker to batch-analyze your images
- Link the executable workflow in your publication