Supplementary Section 4. Creating a BIA workflow and adding it to a BIAFLOWS instance

The procedure to package a workflow and add it to a BIAFLOWS instance is described in details in this section. Users willing to get help can contact biaflows@neubias.org.

Sample workflows running in FIJI, ICY, CellProfiler, ilastik, Vaa3D, Python and Jupyter notebooks can be found in this GitHub repository: https://github.com/neubias-wg5.

Introduction

BIAFLOWS workflows are Docker images encapsulating a complete execution environment together with a workflow addressing a BIA Problem from an existing **BIAFLOWS** Problem Class (see **Problem Class ground truth annotations formats and reported metrics**).

These Docker images can be versioned and compiled automatically online. Once configured, a BIAFLOWS instance automatically fetches new workflow versions and makes them available from BIAFLOWS user interface.

Before adding new workflows to an existing instance, it is highly advised to first watch tutorial videos and explore BIAFLOWS online instance (https://biaflows.neubias.org/) to get familiar with the platform. You can refer to the developer guide to get more details (**Installing and populating BIAFLOWS locally**) more details.

BIA workflow requirements

BIAFLOWS workflows must:

- Run headless from command line
- Take an input folder of TIFF (2D) or single file OME-TIFF (C,Z,T) images
- Expose functional parameters and parse them from command line call
- Export results to an output folder in a format specified for the Problem Class (see **Problem Class ground truth annotations formats and reported metrics**).

The workflow and its software execution environment are fully defined from a set of 4 files:

- A Dockerfile configuring software execution environment (OS, libraries, software...)
- The workflow executable or, more commonly, a script running on a BIA platform
- A Python script (wrapper.py), the Docker image entry point sequencing all operations
- A descriptor (descriptor.json) specifying workflow parameters and default values.

Step 1. Create a workflow GitHub repository

Create a workflow repository in a GitHub source trusted by the BIAFLOWS instance you plan to add the workflow to; for BIAFLOWS online instance: https://github.com/Neubias-WG5. Workflow repository names should start by a fixed prefix (e.g. **W_** for BIAFLOWS online instance).

Step 2. Add the 4 required files to the workflow repository

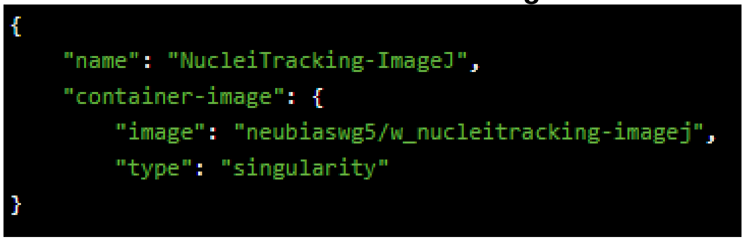
It is highly advised to import these files from existing, similar, workflow repositories (e.g. from https://github.com/Neubias-WG5). Follow these guidelines:

Descriptor: a descriptor from the problem class you target (e.g. Object Segmentation)
DockerFile: a DockerFile configuring the BIA platform you target (e.g ImageJ)
Wrapper script: a wrapper from the problem class **and** the BIA platform you target
Workflow: Your own code.

These BIA platforms and workflow types are already available: ImageJ macro, ImageJ Python script, ICY protocol, CellProfiler pipeline, Octave script, ilastik pipeline, Vaa3D plugin, Python 2.X or 3.X script.

Step 3. Update specific Descriptor sections

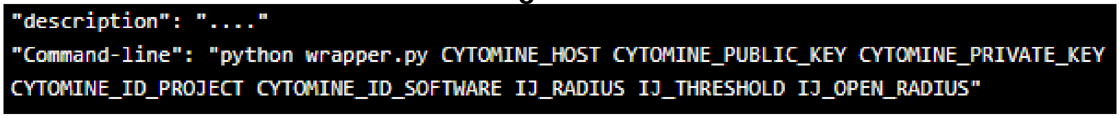
Workflow and associated Docker image names



```
{
    "name": "NucleiTracking-ImageJ",
    "container-image": {
        "image": "neubiaswg5/w_nucleitracking-imagej",
        "type": "singularity"
}
```

Update *name* to match the name of your workflow GitHub repository (without prefix).
Update *image* to match the name of your workflow GitHub repository (lower case only)

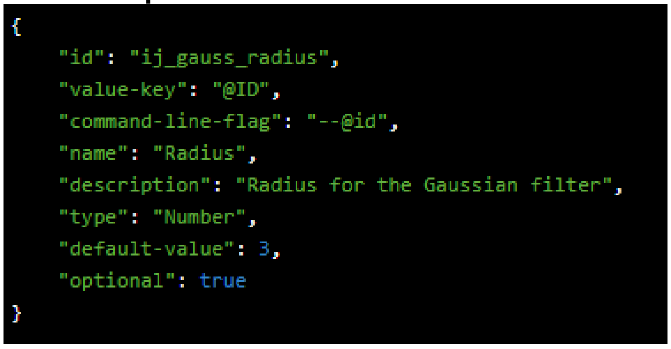
Command line call of the Docker image



```
"description": "...."
"Command-line": "python wrapper.py CYTOMINE_HOST CYTOMINE_PUBLIC_KEY CYTOMINE_PRIVATE_KEY
CYTOMINE_ID_PROJECT CYTOMINE_ID_SOFTWARE IJ_RADIUS IJ_THRESHOLD IJ_OPEN_RADIUS"
```

Description: Update workflow description
Command-line: Update parameter list (here last 3 arguments)

Workflow parameter sections



```
{
    "id": "ij_gauss_radius",
    "value-key": "@ID",
    "command-line-flag": "--@id",
    "name": "Radius",
    "description": "Radius for the Gaussian filter",
    "type": "Number",
    "default-value": 3,
    "optional": true
}
```

Update / add as many parameter sections as required to match parameter list from command line call.

id: should match the parameter name of the command line call (lower case)
name: parameter name as it will appear in BIAFLOWS dialog box

description: context help in BIAFLOWS dialog box
type: String or Number
default-value: the default value of the parameter in BIAFLOWS dialog box.

Step 4. Update DockerFile

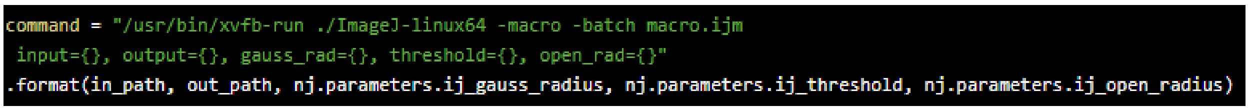
Update the line copying the workflow from the GitHub repository to the workflow Docker image, for instance:

```
ADD NucleiTracking.ijm /fiji/macros/macro.ijm
```

If necessary, add commands to add additional libraries/plugins to execution environment.

Step 5. Update wrapper script

Update workflow command line call in wrapper.py.

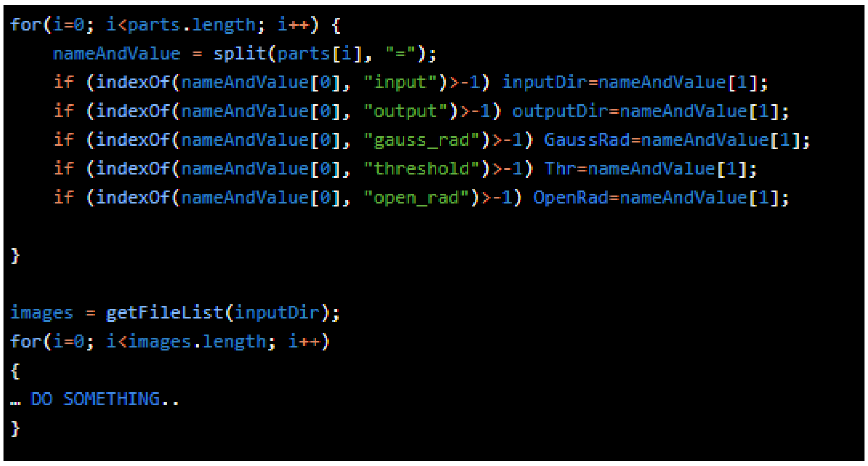


```
command = "/usr/bin/xvfb-run ./ImageJ-linux64 -macro -batch macro.ijm
 input={}, output={}, gauss_rad={}, threshold={}, open_rad={}"
.format(in_path, out_path, nj.parameters.ij_gauss_radius, nj.parameters.ij_threshold, nj.parameters.ij_open_radius)
```

Typically you need to update/add parameters to match the parameters previously defined in the JSON descriptor (Step 2).

Step 6. Adapt your workflow script

Adapt your workflow script to fulfill workflow requirements and read parameters from command line. For instance for an ImageJ macro:

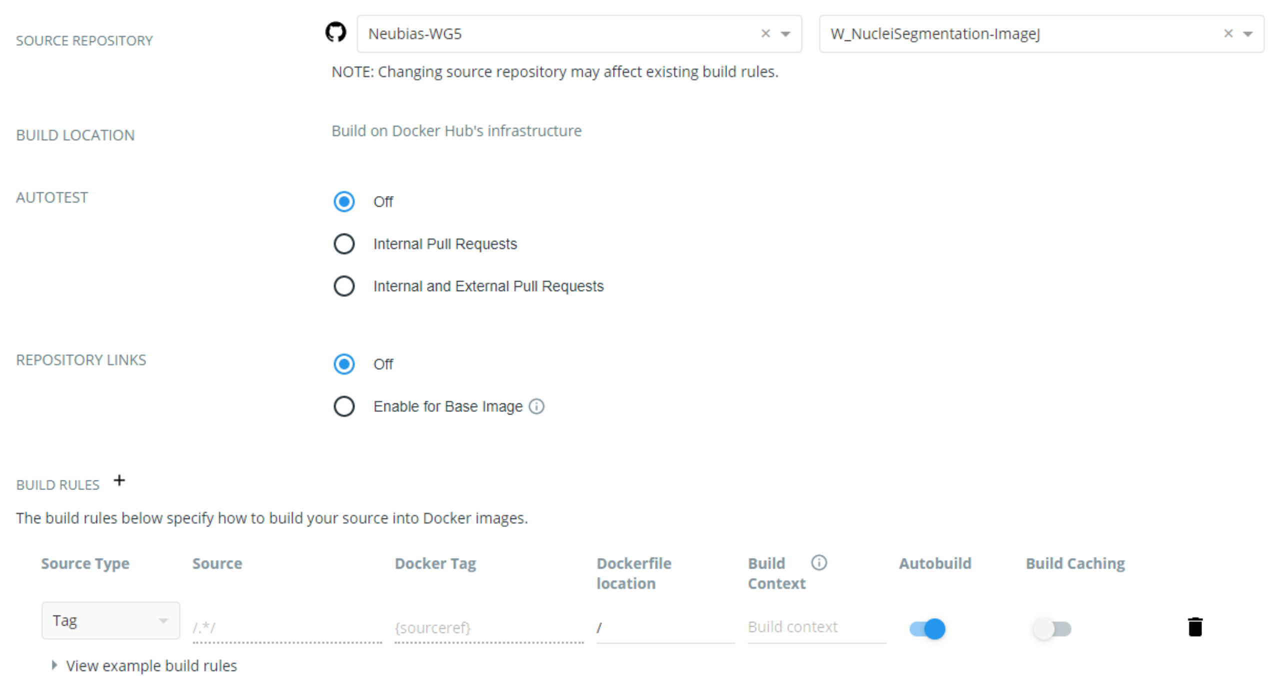


```
for(i=0; i<parts.length; i++) {
    nameAndValue = split(parts[i], "=");
    if (indexOf(nameAndValue[0], "input")>-1) inputDir=nameAndValue[1];
    if (indexOf(nameAndValue[0], "output")>-1) outputDir=nameAndValue[1];
    if (indexOf(nameAndValue[0], "gauss_rad")>-1) GaussRad=nameAndValue[1];
    if (indexOf(nameAndValue[0], "threshold")>-1) Thr=nameAndValue[1];
    if (indexOf(nameAndValue[0], "open_rad")>-1) OpenRad=nameAndValue[1];

}

images = getFileList(inputDir);
for(i=0; i<images.length; i++)
{
… DO SOMETHING..
}
```

Step 7. Create Docker image in DockerHub

Sign in to DockerHub and create a new public repository. The repository name must match the container-image name you used in Step 2.

Step 8. Link repository to workflow GitHub repository and configure workflow Docker image automated build according to the following example:



SOURCE REPOSITORY: Neubias-WG5 / W_NucleiSegmentation-ImageJ

NOTE: Changing source repository may affect existing build rules.

BUILD LOCATION: Build on Docker Hub's infrastructure

AUTOTEST: [✓ Off] Internal Pull Requests / Internal and External Pull Requests

REPOSITORY LINKS: [✓ Off] Enable for Base Image

BUILD RULES +

The build rules below specify how to build your source into Docker images.

Source Type	Source	Docker Tag	Dockerfile location	Build Context	Autobuild	Build Caching
Tag	/.*/	{sourceref}	/	Build context	On	Off

View example build rules

Step 9. Trigger a workflow release

Trigger a release from GitHub workflow repository. Using a version tag such as 0.1 or 1.0.

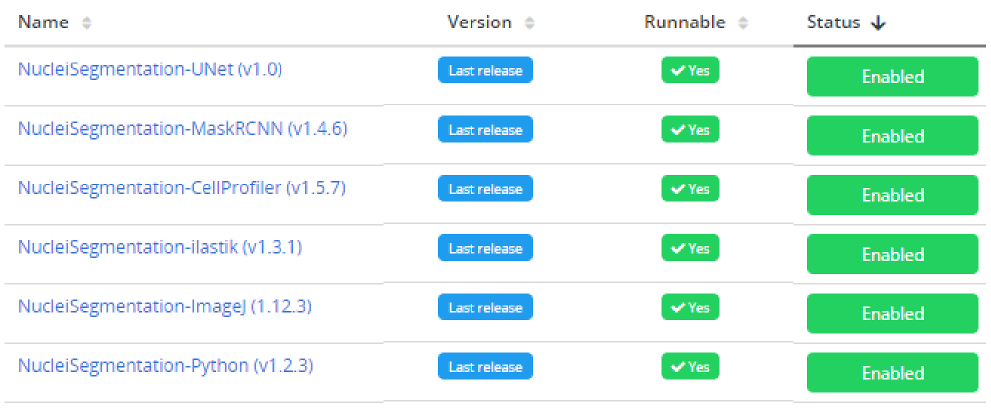
Step 10. Workflow Docker image build

Check from DockerHub that the workflow Docker image has built successfully. If not, parse the log and fix the issue by modifying DockerFile and retriggering a new release.

Step 11. Add workflow to BIAFLOWS problem

Once the workflow Docker image is built, BIAFLOWS should quickly fetch the image and make it available (possibly after up to 15 minutes). Sign in as administrator to BIAFLOWS and browse to the Problem you want to add the workflow to, then click on the Configuration icon.

Search for the workflow (recently added workflows are on top of the list) and enable it. You can also disable older versions if this is an update to an existing workflow.

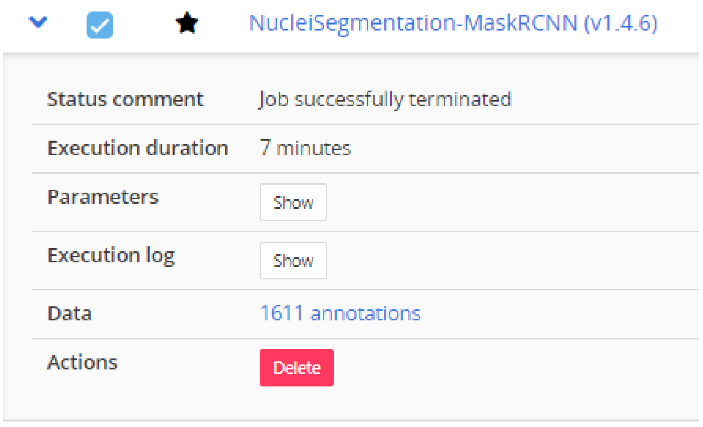


Name	Version	Runnable	Status
NucleiSegmentation-UNet (v1.0)	Last release	✓ Yes	Enabled
NucleiSegmentation-MaskRCNN (v1.4.6)	Last release	✓ Yes	Enabled
NucleiSegmentation-CellProfiler (v1.5.7)	Last release	✓ Yes	Enabled
NucleiSegmentation-ilastik (v1.3.1)	Last release	✓ Yes	Enabled
NucleiSegmentation-ImageJ (1.12.3)	Last release	✓ Yes	Enabled
NucleiSegmentation-Python (v1.2.3)	Last release	✓ Yes	Enabled

Step 12. Run the workflow

Test the workflow by running it from BIAFLOWS/*Workflow runs* (requires execution rights).

If execution fails, read the execution log, adjust the code and trigger a new release.



Status comment	Job successfully terminated
Execution duration	7 minutes
Parameters	Show
Execution log	Show
Data	1611 annotations
Actions	Delete