[1. What are plugins? 3](#_Toc50750388)

[1.1. Types of Plugins 3](#_Toc50750389)

[2. How do they work? 3](#_Toc50750390)

[2.1. Overview 3](#_Toc50750391)

[2.2. Installation 4](#_Toc50750392)

[1.1.1. With SynBioHub 4](#_Toc50750393)

[1.1.1.1. Common problems 4](#_Toc50750394)

[1.1.1.2. Adding plugins in the admin panel 5](#_Toc50750395)

[1.1.2. Standalone Docker Container 5](#_Toc50750396)

[1.1.1.3. Common Problems 5](#_Toc50750397)

[1.1.3. Stand alone 6](#_Toc50750398)

[1.1.1.4. Python Flask Servers 6](#_Toc50750399)

[1.1.1.5. Javascript Node.js with Express Servers 6](#_Toc50750400)

[3. Writing Plugins 7](#_Toc50750401)

[3.1. Naming convention 7](#_Toc50750402)

[3.2. Basic App 7](#_Toc50750403)

[1.1.4. Visual 8](#_Toc50750404)

[1.1.1.6. Status 8](#_Toc50750405)

[Recieves 8](#_Toc50750406)

[Returns 8](#_Toc50750407)

[1.1.1.7. Evaluate 8](#_Toc50750408)

[Recieves 8](#_Toc50750409)

[Returns 8](#_Toc50750410)

[1.1.1.8. Run 8](#_Toc50750411)

[Recieves 8](#_Toc50750412)

[Returns 9](#_Toc50750413)

[1.1.1.9. Examples 9](#_Toc50750414)

[Python 9](#_Toc50750415)

[Javascript 9](#_Toc50750416)

[3.3. Submit 9](#_Toc50750417)

[1.1.1.10. Status 9](#_Toc50750418)

[Recieves 9](#_Toc50750419)

[Returns 9](#_Toc50750420)

[1.1.1.1. Evaluate 9](#_Toc50750421)

[Recieves 9](#_Toc50750422)

[Returns 10](#_Toc50750423)

[1.1.1.2. Run 11](#_Toc50750424)

[Recieves 11](#_Toc50750425)

[Returns 12](#_Toc50750426)

[1.1.1.3. Examples 12](#_Toc50750427)

[Python 12](#_Toc50750428)

[Javascript 12](#_Toc50750429)

[3.4. Download 12](#_Toc50750430)

[1.1.1.4. Status 12](#_Toc50750431)

[Recieves 12](#_Toc50750432)

[Returns 12](#_Toc50750433)

[1.1.1.5. Evaluate 13](#_Toc50750434)

[Recieves 13](#_Toc50750435)

[Returns 13](#_Toc50750436)

[1.1.1.1. Run 13](#_Toc50750437)

[Recieves 13](#_Toc50750438)

[Returns 14](#_Toc50750439)

[1.1.1.2. Examples 14](#_Toc50750440)

[Python 14](#_Toc50750441)

[Javascript 14](#_Toc50750442)

[3.5. Writing an app by copying the examples 14](#_Toc50750443)

[4. With Virtual Environment 14](#_Toc50750444)

[4.1. For Anaconda: 14](#_Toc50750445)

[5. Without Virtual Environment 15](#_Toc50750446)

[6. Stackoverflow link 15](#_Toc50750447)

[6.1. Best practices 15](#_Toc50750448)

[6.2. Advanced possibilities 15](#_Toc50750449)

[6.3. Dockerising 15](#_Toc50750450)

[1.1.5. Simple overview of docker 15](#_Toc50750451)

[1.1.6. Steps for dockerising a plugin 16](#_Toc50750452)

[1.1.7. Useful commands for docker debug 17](#_Toc50750453)

[7. Genetic Logic Lab Synbiohub plugins 17](#_Toc50750454)

# What are plugins?

Plugins are modular stand-alone additions to SynBioHub. The function in a similar way to browser extensions in that they can be installed separate from the browser/SynBioHub and provide additionally ‘custom’ functionality to the browser/SynBioHub experience despite having a completely separate code base to the browser/SynBioHub they seem integrated to the user.

## Types of Plugins

Currently there are three types of plugins Submit, Visual, and Download.

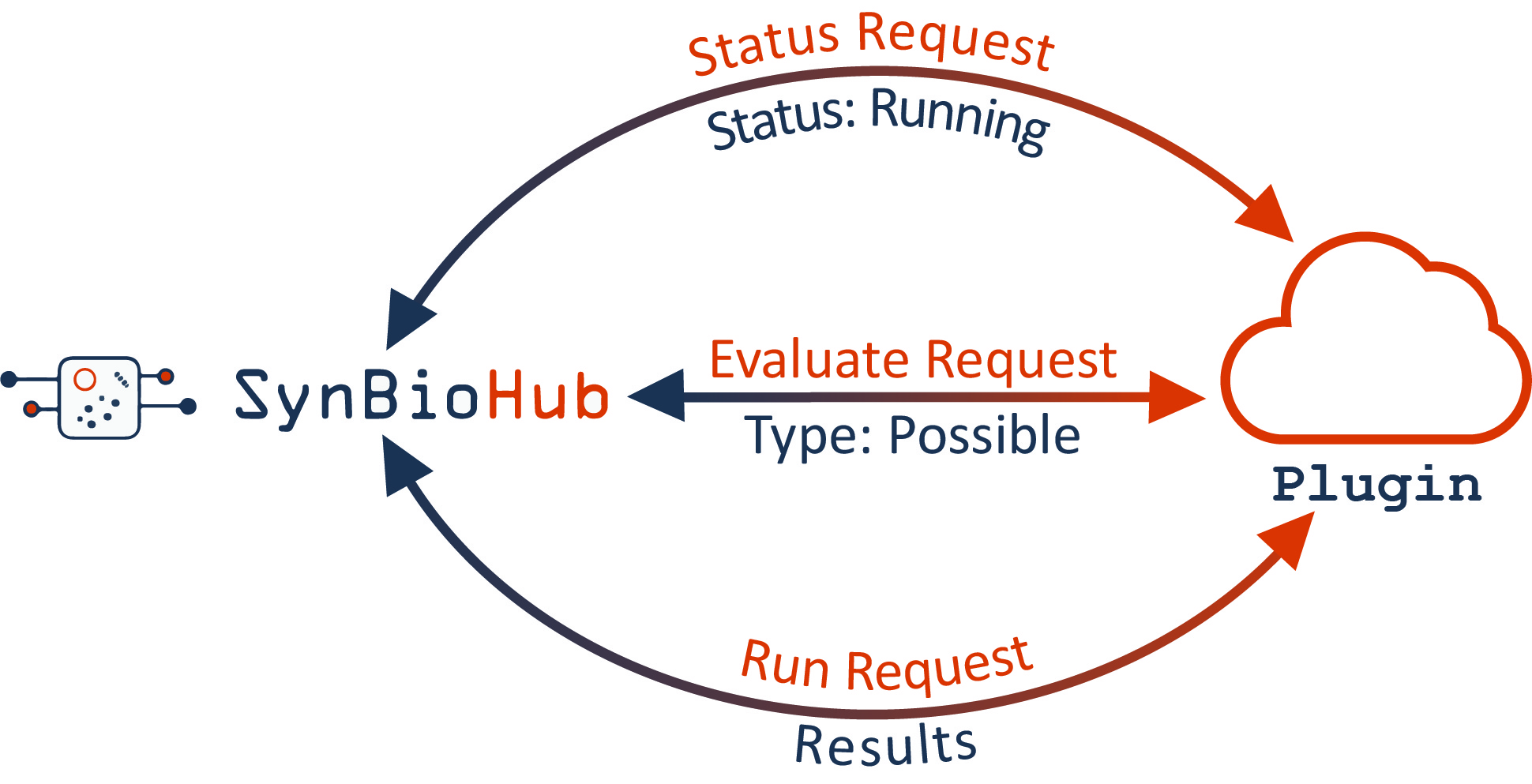
Submit plugins are available to use from the submit endpoint (LINK TO SUBMIT DOCUMENTATION). They work by taking in the file that is uploaded in the submit and processing it to return SBOL to the SynBioHub endpoint.

Visual plugins are available on all ‘endpoint’ pages, for example pages for components, sequences, activities, etc. Visual plugins return html to be displayed on the page.

Download plugins are available on all ‘endpoint’ pages, for example pages for components, sequences, activities, etc. Download plugins return some kind of file which is downloaded by the user.

# How do they work?

## Overview



All plugins are servers (a piece of software in a network that is used to provide services (such as access to files or the routing of e-mail) to other pieces of software or hardware (clients) in the network). All plugins have at least 3 endpoints: Status, Evaluate, and Run. SynBioHub sends a status request to the status endpoint, if the response is that the plugin is up and running SynBioHub then sends an evaluate request. The evaluate request tests whether the plugin can handle the data that SynBioHub wishes to send. If the plugin responds that it can handle the data SynBioHub then sends the data to the run endpoint and uses the final results in the appropriate manner for the plugin type.

Note that plugins are asynchronous meaning that the full SynBioHub page can load without having to wait for the plugin to have returned results. Then when the results are returned SynBioHub substitutes the received information for the placeholder it had used when loading the page initially.

## Installation

As the plugins are standalone servers they do not have to be used in conjunction with SynBioHub. Though plugins have been developed to interface directly with SynBioHub it is possible to ‘stand them up’ (install them) and then interact with them via the API (from the command line or via a GUI interface such as Postman <https://www.postman.com/>). Different methods of installation are discussed below. For the less programmatically confident we suggest just using plugins with SynBioHub.

### With SynBioHub

To install plugins with synbiohub the easiest way is to use Docker Compose’s [multiple file](https://docs.docker.com/compose/reference/overview/#specifying-multiple-compose-file) capabilities. (Note you will need to install docker desktop and open it as an administrator before the commands bellow can be carried out)

1. Open terminal
2. Change to a directory where you are happy to pull the synbiohub docker files
3. git clone https://github.com/synbiohub/synbiohub-docker
4. docker-compose --f ./synbiohub-docker/docker-compose.yml -f ./synbiohub-docker/docker-compose.explorer.yml -f ./synbiohub-docker/docker-compose.**<Plugin 1 File Name>**.yml -f ./synbiohub-docker/docker-compose.**<Plugin 2 File Name>**.yml up

Note that all plugins are added before the up and each is preceeded by -f . For example, to run the configuration with the VisualIgem plugins and the VisualSeqviz plugin run:

docker-compose --f ./synbiohub-docker/docker-compose.yml -f ./synbiohub-docker/docker-compose.explorer.yml -f ./synbiohub-docker/docker-compose.pluginVisualIgem.yml -f ./synbiohub-docker/docker-compose.pluginVisualSeqviz.yml up

A full list of plugin file names and their descriptions can be found here: <https://synbiohub.github.io/synbiohub-docker/#plugins>

#### Common problems

* The plugins might try to use ports that are already in use. If upon running the above command and then using `docker ps` not all of the ‘images’ that you expect are present then try `docker ps –a`. If you now see the image(s) you were missing with the status ‘created’ it suggests that they have a port clash. If this is the case you can go in the docker-compose file for the appropriate plugin and change the port number (the first four digit number under “ports”). Then try running step 4 of the above commands again.
* If you have locally installed synbiohub using the above it might cause issues in some plugins that will receive urls etc that look like: <http://localhost:7777/xxxxxxx>. Localhost in docker containers refers to the container itself. The problem is explained further here (https://medium.com/it-dead-inside/docker-containers-and-localhost-cannot-assign-requested-address-6ac7bc0d042b) On Linux this problem is solvable (<https://stackoverflow.com/questions/31324981/how-to-access-host-port-from-docker-container> and <https://docs.docker.com/network/host/>).

#### Adding plugins in the admin panel

After getting the plugins up and running you still have to add them to the admin panel in SynBioHub. Navigate to the plugin section of the admin panel. Choose the appropriate section (rendering, submit, or download) and give the plugin a descriptive name. Then in the url section there are several different options depending on exactly how the plugin was brought up:

* If plugins were brought up using docker compose:
  + ‘http://localhost:**<port>**/’ e.g. ‘http://localhost:8093/’ for the Excel Submit Library plugin
  + Alternative: ‘http://**<docker container name>**:5000/’ e.g. ‘http://synbiohub-docker\_pluginSubmitExcelLibrary\_1:5000/’ for the Excel Submit Library plugin
  + Alternative (not recommended): ‘http://**<synbiohub docker network ip address>**:**<port>** /’ e.g. <http://172.18.0.1:8093/> for the Excel Submit Library plugin when the synbiohub-docker network is 172.18.0.1
    - Find out the synbiohub docker network ip by by typing into the terminal `docker inspect synbiohub-docker\_default` and the address following ‘Gateway’ in the print out is the ip address you want. Note that this will change every time you run the docker compose command
* If plugins are hosted elsewhere on a public server:
  + Simply use the url to access them e.g.: <https://seqviz.synbiohub.org/> for the seqviz plugin (this method is also possible if you have named and exposed internal ports from your docker compose (this requires caddy))

### Standalone Docker Container

If you just want to bring up the plugin as a stand-alone Docker container you can simply pull the image from docker hub. First make sure you have docker desktop installed and open and running (run as an administrator). Then in the terminal run `docker run --publish **<publish port>**:5000 --detach --name **<nickname of your choice>** synbiohub/**<docker image name>**`

e.g: `docker run --publish 8093:5000 --detach --name submit-plug synbiohub/plugin-submit-excel-library:snapshot` for the excel library submit plugin.

We suggest using the same publish ports as are used when using docker compose. The list of those ports can be found here: <https://synbiohub.github.io/synbiohub-docker/#plugins>

#### Common Problems

* Localhost. If the plugin is being run as a docker container that localhost may cause problems as localhost is considered within the container and won’t point to the localhost on your computer. The problem is explained further here (https://medium.com/it-dead-inside/docker-containers-and-localhost-cannot-assign-requested-address-6ac7bc0d042b) On Linux this problem is solvable (<https://stackoverflow.com/questions/31324981/how-to-access-host-port-from-docker-container> and <https://docs.docker.com/network/host/>).

### Stand alone

Plugins can also be run without using docker. How this is done depends on the language and packages used to create the server. We provide documentation for two of the most common server choices: python using flask and javascript using express.

#### Python Flask Servers

NOTE: You might have to add pandas to the requirements if you are running the plugins as standalone (this is not needed if you are running the plugin via docker as we use a base image which contains pandas preinstalled as it is a nightmare to install via docker on a just python base image)

* Anaconda
  + Open terminal
  + Git clone repository (e.g. `git clone https://github.com/SynBioHub/Plugin-Submit-Excel-Library.git`)
  + Change into the repository folder
  + Open anaconda prompt and carry out the rest of the commands there
  + conda install --file requirements.txt
  + `set FLASK\_APP=app.py`
  + OPTIONAL: set FLASK\_ENV=development
  + flask run --port 5000 (the -- port 5000 is optional and can be any port number)
  + The server should then be accessible at http://localhost:5000/status
* Python
  + Mac/Linux
    - Open terminal
    - Git clone repository (e.g. `git clone https://github.com/SynBioHub/Plugin-Submit-Excel-Library.git`)
    - Change into the repository folder
    - `pip install -r requirements.txt`
    - `export FLASK\_APP=app`
    - `flask run`
    - The server should then be accessible at http://localhost:5000/status
  + Windows
    - Open terminal
    - Git clone repository (e.g. `git clone https://github.com/SynBioHub/Plugin-Submit-Excel-Library.git`)
    - Change into the repository folder
    - `pip install -r requirements.txt`
    - Powershell: $env:FLASK\_APP = "hello", Command: set FLASK\_APP=hello
    - `flask run`
    - The server should then be accessible at http://localhost:5000/status

For more information see: https://flask.palletsprojects.com/en/1.1.x/cli/

#### Javascript Node.js with Express Servers

* Open terminal
* Git clone repository (e.g. `git clone https://github.com/SynBioHub/ Plugin-Submit-Test-js.git`)
* Change into the repository folder
* `npm install` (installs all requirements requires npm to be installed: https://nodejs.org/en/download/)
* node app.js
* The server should then be accessible at http://localhost:5000/status

# Writing Plugins

## Naming convention

For the github repositories the naming convention for plugin repositories is: Plugin-<Type>-<Description>-<Test>-<language>

Note that words are separated by a dash (-) and every word but the program language is Proper case (intial letter and no further letters capitalized).

For Type it is: Submit, Visual, Download

For Test: add the Test to the name if it is a framework plugin simply showing off how to write a particular plugin type

For language:

* Python: no addition
* Javascript: js

Examples:

* Plugin-Submit-Excel-Library
* Plugin-Download-Test-js
* Plugin-Visual-Component-Use

The same name is then used for both the docker-compose and the docker image name:

E.g.:

* Repository: Plugin-Visual-Component-Use
* Docker Compose File: docker-compose.pluginVisualComponentUse.yml (note the plugin is lowercase and then the rest of the words are capitalized but no spaces or dashes)
* Docker Image Name: synbiohub/plugin-visual-component-use:snapshot (note no capitals in the docker image name)

## Basic App

In this section exactly what the server receives and the format of its response is defined. Below is a section about writing a server/plugin by simply copying and editing the example type.

### Visual

#### Status

##### Recieves

Nothing is received (it is a simple get request)

##### Returns

If it is up and running it should return a 200 status to a get request

#### Evaluate

##### Recieves

1. type: The RDF type of the top-level object

Common RDF types are:

'Activity', 'Agent', 'Association', 'Attachment', 'Collection', 'CombinatorialDerivation', 'Component', 'ComponentDefinition', 'Cut', 'Experiment', 'ExperimentalData', 'FunctionalComponent','GenericLocation', 'Implementation', 'Interaction', 'Location', 'MapsTo', 'Measure', 'Model', 'Module', 'ModuleDefinition', 'Participation', 'Plan', 'Range', 'Sequence', 'SequenceAnnotation', 'SequenceConstraint', 'Usage', 'VariableComponent'

Example 1:

{'type': 'Component'}

##### Returns

Return a status of 200 if the type is acceptable and a 4xx status if it isn’t

#### Run

##### Recieves

1. complete\_sbol: the single-use URL for the complete object to operate on
2. shallow\_sbol: the single-use URL for a summarized or truncated view of the object
3. genbank: the single-use URL for the Genbank of the object (Note this will be a blank website for all types other than Component)
4. top\_level: the top-level URL of the SBOL object
5. instanceUrl: the top-level URL of the synbiohub instance
6. size: a number representing an estimate of the size of the object, probably triple count
7. type: The RDF type of the top-level object

Common RDF types are:

'Activity', 'Agent', 'Association', 'Attachment', 'Collection', 'CombinatorialDerivation', 'Component', 'ComponentDefinition', 'Cut', 'Experiment', 'ExperimentalData', 'FunctionalComponent','GenericLocation', 'Implementation', 'Interaction', 'Location', 'MapsTo', 'Measure', 'Model', 'Module', 'ModuleDefinition', 'Participation', 'Plan', 'Range', 'Sequence', 'SequenceAnnotation', 'SequenceConstraint', 'Usage', 'VariableComponent'

Example 1:

{'complete\_sbol': 'https://dev.synbiohub.org/public/igem/BBa\_E0040/1/sbol',

'shallow\_sbol': 'https://dev.synbiohub.org/public/igem/BBa\_E0040/1/sbolnr',

'genbank': 'https://dev.synbiohub.org/public/igem/BBa\_E0040/1/gb',

'top\_level': 'https://synbiohub.org/public/igem/BBa\_E0040/1',

'size': 5,

'type': 'Component',

'instanceUrl': 'https://dev.synbiohub.org/'}

Example 2:

{'complete\_sbol': 'https://synbiohub.org/public/igem/BBa\_E0240\_sequence/1/sbol', 'shallow\_sbol': 'https://synbiohub.org/public/igem/BBa\_E0240\_sequence/1/sbolnr', 'genbank': 'https://synbiohub.org/public/igem/BBa\_E0240\_sequence/1/gb', 'top\_level': 'https://synbiohub.org/public/igem/BBa\_E0240\_sequence/1', 'size': 2, 'type': 'Sequence', 'instanceUrl': 'https://synbiohub.org/'}

##### Returns

The plugin should respond with an HTML page to be rendered in-frame on the corresponding SynBioHub page.

#### Examples

##### Python

A full example of a visual plugin can be found at: https://github.com/SynBioHub/Plugin-Visual-Test

##### Javascript

A full example of a visual plugin can be found at: https://github.com/SynBioHub/Plugin-Visual-Test-js

## Submit

#### Status

##### Recieves

Nothing is received (it is a simple get request)

##### Returns

If it is up and running it should return a 200 status to a get request

#### Evaluate

##### Recieves

A dictionary with entries:

* {‘manifest’
  + {‘files’
    - [List of dictionaries (one for every file). Each dictionary has the keys:
      * ‘url’ the single-use URL for the file submitted
      * ‘filename’ the encrypted file name (with correct extension)
      * ‘type’ the mime of the file (see https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics\_of\_HTTP/MIME\_types/Common\_types)

Example 1:

{"manifest": {"files":[

  {"url": "http://synbiohub.org/expose/b41e63d6-10f4-4cac-b1c8-285f71156b56", "filename": "asdfasdf.xls", "type": "application/vnd.ms-excel"},

  {"url": "http://synbiohub.org/expose/jkl9d8s7ufjqhoer8u709s", "filename": "file\_name1.dna", "type": "application/xml"},

  {"url": "http://synbiohub.org/expose/basdf-11230948f4-12344cac", "filename": "file\_name2.xml", "type": "application/xml"},

  {"url": "http://synbiohub.org/expose/09uj2k3j0", "filename": "file\_name3.xml", "type": "application/xml"},

  {"url": "http://synbiohub.org/expose/asdfasdf56", "filename": "file\_name4.xml", "type": "application/xml"}]}}

Example 2:

{"manifest": {"files":[

  {"url": "http://synbiohub.org/expose/93c25d15-9fe2-4862-b602-ddbdac2c4333", "filename": "mbr5wW6CIseoq-0Y4MkM8DTM.xml", "type": "application/xml"}]}}

##### Returns

* {‘manifest’:[
  + Dictionaries each of which contains
    - ‘filename’: the encrypted file name (with correct extension) that matches the original filename sent in the manifest from synbiohub
    - ‘requirement’: a number which indicates whether or not the file can be used the numbers mean: 2-file will be converted to sbol, 1-file will be used to convert other files to sbol, 0-file cannot be handled/is not useful

Example 1:

{"manifest": [

        {

            "filename": " asdfasdf.xls ",

            "requirement": 2

        },

        {

            "filename": "file\_name1.dna",

            "requirement": 1

        },

        {

            "filename": "file\_name2.xml",

            "requirement": 0

        },

        {

            "filename": "file\_name3.xml",

            "requirement": 0

        },

        {

            "filename": "file\_name4.xml",

            "requirement": 0

        }

    ]

}

Example 2:

{

    "manifest": [

        {

            "filename": " mbr5wW6CIseoq-0Y4MkM8DTM.xml ",

            "requirement": 2

        }

    ]

}

#### Run

##### Recieves

A dictionary with entries:

* {‘manifest’
  + {‘files’
    - [List of dictionaries (one for every file). Each dictionary has the keys:
      * ‘url’ the single-use URL for the file submitted
      * ‘filename’ the encrypted file name (with correct extension)
      * ‘type’ the mime of the file (see <https://developer.mozilla.org/en-US/docs/Web/HTTP/Basics_of_HTTP/MIME_types/Common_types>)
    - ‘instanceUrl’ the top-level URL of the synbiohub instance

Example 1:

{'manifest': {'files': [{'filename': '0WaHXIKZD10gRGdC8U7weHWC.html', 'type': 'text/html', 'url': 'https://dev.synbiohub.org/expose/e9665a54-dbdd-485e-a20b-574e46412fc4'}]}, 'instanceUrl': 'https://dev.synbiohub.org/'}

Example 2:

{"manifest": {"files":[{"url": "https://synbiohub.org/expose/kl98ahnj0fa9u03945r",

                "filename": "oijwpjrpokk98um098m.xlsx",

                "type": "application/vnd.openxmlformats-officedocument.spreadsheetml.sheet"},

{"url": "https://synbiohub.org/98ja0s98dufjml.dna",

                "filename": "as9d8j0asd9j23fc.dna",

                "type": " application/xml "}]},

                "instanceUrl": "http://localhost:7777/"}

##### Returns

A zip file which contains the generated SBOL files and a file called manifest.json which contains a json response manifest of the form:

* {‘results’
  + [List of dictionaries (one for every file). Each dictionary has the keys:
    - ‘filename’ name of the file within the zipfile
    - ‘sources’ a list of filenames received from synbiohub that were used to generate the file

Example 1:

{'results': [{'filename': '0WaHXIKZD10gRGdC8U7weHWC.html.converted', 'sources': ['0WaHXIKZD10gRGdC8U7weHWC.html ']}]}

Example 2:

{'results': [{'filename': oijwpjrpokk98um098m.xlsx.converted', 'sources': [‘oijwpjrpokk98um098m.xlsx']}, {'filename': ' as9d8j0asd9j23fc.dna.converted', 'sources': [' as9d8j0asd9j23fc.dna’]}]}

NOTE: list of sources could contain more than one file (if any of the files in the evaluate endpoint were flagged as 1 they might be added to the sources for one of the other file conversions)

#### Examples

##### Python

A full example of a submit plugin can be found at: https://github.com/SynBioHub/Plugin-Submit-Test

##### Javascript

A full example of a submit plugin can be found at: <https://github.com/SynBioHub/Plugin-Submit-Test-js>

## Download

#### Status

##### Recieves

Nothing is received (it is a simple get request)

##### Returns

If it is up and running it should return a 200 status to a get request

#### Evaluate

##### Recieves

* 1. type: The RDF type of the top-level object

Common RDF types are:

'Activity', 'Agent', 'Association', 'Attachment', 'Collection', 'CombinatorialDerivation', 'Component', 'ComponentDefinition', 'Cut', 'Experiment', 'ExperimentalData', 'FunctionalComponent','GenericLocation', 'Implementation', 'Interaction', 'Location', 'MapsTo', 'Measure', 'Model', 'Module', 'ModuleDefinition', 'Participation', 'Plan', 'Range', 'Sequence', 'SequenceAnnotation', 'SequenceConstraint', 'Usage', 'VariableComponent'

Example 1:

{'type': 'Component'}

##### Returns

Return a status of 200 if the type is acceptable and a 4xx status if it isn’t

#### Run

##### Recieves

1. complete\_sbol: the single-use URL for the complete object to operate on
2. shallow\_sbol: the single-use URL for a summarized or truncated view of the object
3. genbank: the single-use URL for the Genbank of the object (Note this will be a blank website for all types other than Component)
4. top\_level: the top-level URL of the SBOL object
5. instanceUrl: the top-level URL of the synbiohub instance
6. size: a number representing an estimate of the size of the object, probably triple count
7. type: The RDF type of the top-level object

Common RDF types are:

'Activity', 'Agent', 'Association', 'Attachment', 'Collection', 'CombinatorialDerivation', 'Component', 'ComponentDefinition', 'Cut', 'Experiment', 'ExperimentalData', 'FunctionalComponent','GenericLocation', 'Implementation', 'Interaction', 'Location', 'MapsTo', 'Measure', 'Model', 'Module', 'ModuleDefinition', 'Participation', 'Plan', 'Range', 'Sequence', 'SequenceAnnotation', 'SequenceConstraint', 'Usage', 'VariableComponent'

Example 1:

{'complete\_sbol': 'https://dev.synbiohub.org/public/igem/BBa\_E0240/1/sbol', 'shallow\_sbol': 'https://dev.synbiohub.org/public/igem/BBa\_E0240/1/sbolnr', 'genbank': 'https://dev.synbiohub.org/public/igem/BBa\_E0240/1/gb', 'top\_level': 'https://synbiohub.org/public/igem/BBa\_E0240/1', 'size': 39, 'type': 'Component', 'instanceUrl': 'https://dev.synbiohub.org/'}

Example 2:

{'complete\_sbol': 'https:// synbiohub.org/public/igem/BBa\_E0240/1/sbol', 'shallow\_sbol': 'https:// synbiohub.org/public/igem/BBa\_E0240/1/sbolnr', 'genbank': 'https:// synbiohub.org/public/igem/BBa\_E0240/1/gb', 'top\_level': 'https://synbiohub.org/public/igem/BBa\_E0240/1', 'size': 39, 'type': 'Component', 'instanceUrl': 'https:// synbiohub.org/'}

##### Returns

The plugin should respond with an HTTP request and file attachment which represents the object.

#### Examples

##### Python

A full example of a download plugin can be found at: https://github.com/SynBioHub/Plugin-Download-Test

##### Javascript

A full example of a download plugin can be found at: https://github.com/SynBioHub/Plugin-Download-Test-js

## Writing an app by copying the examples

* Copy the relevant repository (make sure it includes the docker actions to automatically build)
* Change the docker image name in release.yml in line 17 and 24
* Ensure the repository has the docker username and docker password as secrets in the repository (will happen automatically if you are creating a repository in the synbiohub project)
* Update the liscence
* Change the read me
* Edit the sections of the code in app.py that say edit within here (should be a section under the evaluate endpoint and under the run endpoint). Can import further functions at the top of the file and do most of the code writing in a second file to leave the structure between plugins as similar as possible
* Create a new requirements.txt or packages.json to match what you now have (packages.json should happen mostly automatically). For requirements.txt there are several methods:

# With Virtual Environment

## For Anaconda:

* pip3 install virtualenv
* python3 -m venv <myenvname>
* conda list -e > requirements.txt

# Without Virtual Environment

Try the pipreqs package (used from anaconda prompt if using anaconda)

# Stackoverflow link

This provides further suggestions and ideas: <https://stackoverflow.com/questions/31684375/automatically-create-requirements-txt>

## Best practices

* Always write files to temporary files or directories (prevents overwriting if you have multiple calls to the same plugin concurrently)
* Never use “file” as a variable name in python (it causes deeper issues)
* Make code modular and comment well
* Adhere to standards and best practices of the language you write in

## Advanced possibilities

* It is possible to use the url to pass parameters to the plugin see (<https://github.com/SynBioHub/Plugin-Visual-Component-Use>)
* It is possible to have files served from the plugin (<https://github.com/SynBioHub/Plugin-Visual-Serve-Test> and <https://github.com/SynBioHub/Plugin-Visual-Serve-Test-js>)

## Dockerising

### Simple overview of docker

Source code is used in docker file which is used to build an image which can be pushed to docker hub (LIKE GITHUB). Image (can have been pulled from dockerhub) can be run on its own or with others like autoheal in a docker compose file.

When an image/docker compose file is run it creates a container.

Caddy (or other webserver like engineX, httpd, apache) listens to https port (443) and converts https to http and routes it to port on which the container is listening which it container thinks is port 80 (http port - generally the default over https, you could map out from a different port depending on what the webserver inside the container thinks it is listening too e.g. synbiohub thinks it listens to port 7777). Container maps its port to host and caddy forwards from 443 to the host port that the container is listening on.

Idea of architecture:

caddy is a webserver which reroutes requests to containers containers may or may not contain images which are also webservers (dockercompse files describe a set of containers which also may or may not contain webservers)

High ports are over 1024 and 'non-reserved' ports. ssh is to 22 (generally).

Key points understand that docker has: images, containers, volumes, and networks (especially important to understand what sticks around and what doesn’t when you try and refresh the containers a very useful command is prune: <https://takacsmark.com/docker-prune/>

### Steps for dockerising a plugin

After having installed docker desktop and making sure it is up and running using an administrator account. Note that the docker website has a tutorial with mor information.

How to dockerise plugin code (things in <> signs should be replaced with your specific variable names)

1. run command window as admin
2. git clone the github repository containing everything into a local directory (git clone <github url>)
3. cd into directory that was just cloned (cd <directory name>)
4. create a file named ".dockerignore" with the contents "Dockerfile" (echo Dockerfile>.dockerignore )
5. create a file named "Dockerfile" (echo >Dockerfile)
6. Edit the Dockerfile to contain the appropriate text (I use notepad++ for this - note for python plugins a requirements.txt is required and for js plugins a package.json file is required)
7. Build your docker image (docker build --tag synbiohub/<descriptive name>:snapshot .) Note the full stop at the end of the command!
8. Run the docker image (docker run --publish <port number>:5000 --detach --name <short name> synbiohub/<descriptive name>:snapshot)
9. check the run was sucessful (docker ps), if not shown there check (docker ps -all) and then check the logs (docker logs <short name>)
10. based on error logs fix the errors, then remove the current container (docker rm <short name>) and repeat steps 7 and 8
11. If docker ps shows your plugin running then navigate to localhost:/status. Can also test the run end point using a program like postman to submit a query and look at the response.
12. If all is well stop the docker container ( docker stop <short name> ) and remove it ( docker rm <short name> )
13. Upload the newly added files (Dockerfile and .dockerignore) to the github repo
14. Add github action to push image to dockerhub (called release.yml in my plugins)
15. under setting>secrets add secrets containing the username and password
16. update the readme and wait for the new image to push to docker hub (you will see a green check next to commit when this has happened)
17. check the uploaded image works by downloading and running it ( docker run --publish 8080:5000 --detach --name <short name> synbiohub/<descriptive name>:snapshot ) and repeating steps 9-12.

### Useful commands for docker debug

* docker system prune -a --volumes
* docker system prune -a
* docker images purge
* docker network inspect synbiohub-docker\_default
* copy docker files to local location to debug: docker cp <name of container>:<path in container> <path on host>
* Enter into a docker file to edit it: docker exec -it synbiohub-docker\_pluginVisualSeqviz\_1 /bin/bash
  + exit by simply typing: exit
  + for synbiohub testing look at plugin.js or view.js in home directory
* docker volume ls
* docker volume prune

# Genetic Logic Lab Synbiohub plugins

* ssh [editors@sbolstandard.org](mailto:editors@sbolstandard.org)
* Password can be found in lab documentation
* cd /opt/synbiohub-docker
* git status check (see it is master)
* sudo git pull (making sure compose files are the latest)
* cd /etc/systemd/system
* sudo emacs synbiohub.service
* add -f docker compose into file
* save: ctrl x ctrl s
* exit: ctrl x ctrl c
* sudo systemctl daemon-reload
* sudo systemctl restart synbiohub
* sudo docker ps (check the container is now listed)
* can check the plugin is okay using: curl localhost:<port>/status
* to give real ip address:
  + cd /etc/caddy
  + sudo emacs Caddyfile
  + copy and paste the igem one and change the name and port number
  + save
  + exit
  + sudo systemctl restart caddy
  + in browser go to azure.microsoft.com: sign in using the credentials from lab documentation
  + go to portal, dns zones
  + go to synbiohub.org
  + set up a new domain: record set, name = seqviz, ip adress: 13.82.19.198, TTL 1 hour, type A, alias record set no.
  + give it a minute to go through then
  + command line: sudo systemctl restart caddy
  + sudo systemctl status caddy
  + q
* IN admin panel on synbiohub:
  + Adding to dev.synbiohub: http://localhost:8095/sankey
  + adding to synbiohub.org: http://synbiohub-docker\_pluginSubmitExcelLibrary\_1:5000/

IF WANTING TO UPDATE AN EXISTING PLUGIN

* ssh [editors@sbolstandard.org](mailto:editors@sbolstandard.org)
* Password from lab documentation
* docker ps
* docker stop relevant\_container
* docker system prune –a (removes all stopped containers)
* sudo systemctl daemon-reload
* sudo systemctl restart synbiohub
* can check the plugin is okay using: curl localhost:<port>/status