
plantit

Release v0.0.2-alpha

Computational Plant Science Lab

Apr 02, 2023

INTRODUCTION

1	About plantit	1
1.1	What is it?	1
1.2	What isn't it?	2
1.3	What can I use it for?	2
2	Quickstart	3
2.1	Use cases	3
2.2	Conceptual model	4
2.2.1	Datasets	4
2.2.2	Agents	4
2.2.3	Workflows	4
2.2.4	Tasks	4
2.2.5	Projects	5
2.3	Submitting tasks	5
2.3.1	Select a workflow	5
2.3.2	Submit to an agent	7
2.3.3	Monitor status	8
2.3.4	Retrieve results	10
3	DIRT Migration	11
3.1	Migrating DIRT datasets	11
4	Workflows	13
4.1	Using workflows	14
4.2	Binding workflows: the plantit.yaml file	14
4.2.1	Required attributes	14
4.2.2	Optional attributes & sections	15
4.3	A simple example	19
4.4	Repository refresh rate	20
5	Datasets	21
5.1	Viewing data	21
5.2	Downloading data	22
5.3	Sharing data	23
5.4	Deleting data	23
6	Agents	25
6.1	Public agents	25
6.2	Integrating a new agent	26
7	Tasks	27

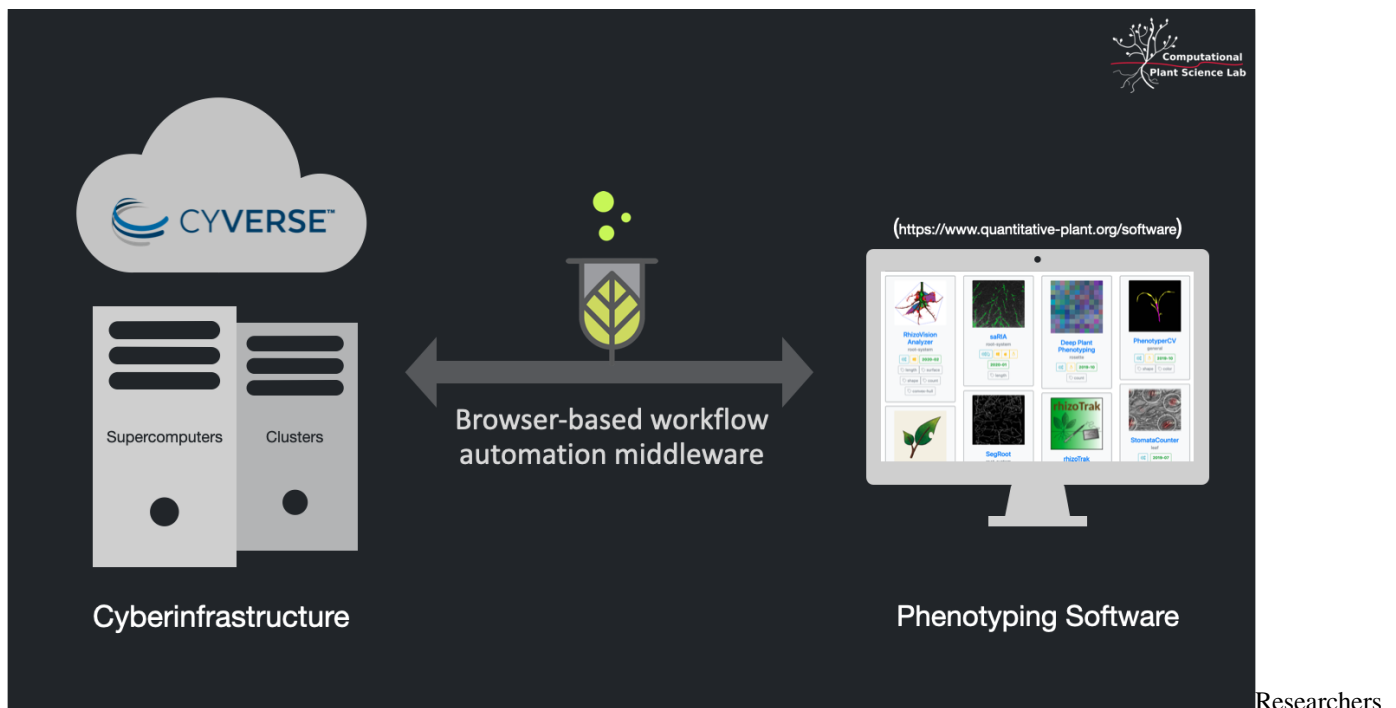
7.1	Task monitoring	27
7.2	Task lifecycle	27
8	Projects	29
9	Examples	31
9.1	Hello world	31
9.2	Parameters	31
9.3	Accessing data	31
10	Architecture	33
10.1	Motivation	33
10.2	Technologies	33
11	Contributions	35
11.1	Configuring a development environment	35
11.2	Command cheatsheet	35
11.2.1	Docker Compose	35
11.2.2	Docker	36
11.2.3	Django	36

ABOUT PLANTIT

- *What is it?*
- *What isn't it?*
- *What can I use it for?*

1.1 What is it?

plantit is a workflow automation tool for computational plant science. It is simultaneously *software-as-a-service* for researchers and a *platform-as-a-service* for programmers & developers.



SaaS for researchers: store, publish, and access data with CyVerse, run (possibly highly parallel) simulations and analyses from a browser.



Configuring workflows

- YAML configuration file (like Travis or GitHub CI)
- Required attributes:

- name
- author
- image
- public
- commands

plantit.yaml

```

1  name: DIRT3D Reconstruction
2  author: Suxing Liu
3  image: docker://computationalplantscience/3d-model-reconstruction
4  public: True
5  commands: python3 /opt/code/cli.py reconstruct "$INPUT" -o $(dirname $INPUT) --gpu $GPU_MODE
6  mount:
7    - /opt/code/vsfm/bin/temp
8    - /opt/code/vsfm/bin/log
9  input:
10   path:
11     kind: directory
12   filetypes:
13     - jpg
14     - png
15  output:
16   path:
17     include:
18       patterns:
19         - nvm
20         - ply
21  logo: Root3D.png
22  gpu: True
23  resources:
24    time: "24:00:00"
25    mem: "20GB"
26    processes: 1
27    cores: 12

```

Developers

Paas for developers: built on GitHub and Docker, add a `plantit.yaml` file to any public GitHub repository to deploy Docker images as Singularity containers on clusters or supercomputers.

1.2 What isn't it?

plantit is none of the following (although it tries to glue these systems together in helpful ways).

- a pipeline orchestrator (e.g., [Snakemake](#), [Nextflow](#), [Luigi](#), [Airflow](#), [Metaflow](#))
- a distributed queue or task scheduler (e.g., [Celery](#) or [Dask](#))
- a batch processing, streaming, or analytics platform (e.g., map-reduce or [Spark](#))
- a container automation system (e.g., [Kubernetes](#))
- a cluster scheduler (e.g., [Torque/Moab](#), [Slurm](#))

1.3 What can I use it for?

plantit is flexible enough to run most container-friendly workloads. If your software can be packaged with Docker and invoked on the command line, plantit can probably run it. That said, plantit is designed primarily for batch processing images in various phenotyping contexts. If you want to do genomics, an established tool like [CoGe](#) or [easyGWAS](#) may be a better fit. Feel free to [get in touch](#) with questions about your use case.

QUICKSTART

- *Use cases*
- *Conceptual model*
 - *Datasets*
 - *Agents*
 - *Workflows*
 - *Tasks*
 - *Projects*
- *Submitting tasks*
 - *Select a workflow*
 - *Submit to an agent*
 - *Monitor status*
 - *Retrieve results*

2.1 Use cases

plantit aims to support two user groups with different concerns and priorities.

- researchers: analyzing data, running models & simulations (**submitting workflows**)
- developers: publishing and maintaining research software (**publishing workflows**)

With plantit the latter group can quickly and easily publish an algorithm to a broader, possibly non-technical user community. In this way it's a continuous deployment tool. It's also a science gateway, hosting plug-and-play algorithms in a web GUI such that any user can leverage XSEDE HPC/HTC clusters for high-throughput phenomics, no programming experience required.

2.2 Conceptual model

plantit has a few fundamental abstractions:

- **Dataset**
- **Agent**
- **Workflow**
- **Task**

A **Dataset** is a set of data objects. A **Workflow** is a containerized research application. A workflow *must* yield a dataset as output and *may* accept one as input (workflows should be designed as functions or generators, *not* for their side effects — ideally, they should have none). An instantiation of a workflow is called a **Task**. An **Agent** is a cluster queue that can run tasks.

2.2.1 Datasets

A **Dataset** is a collection of data objects in the [CyVerse data store](#).

2.2.2 Agents

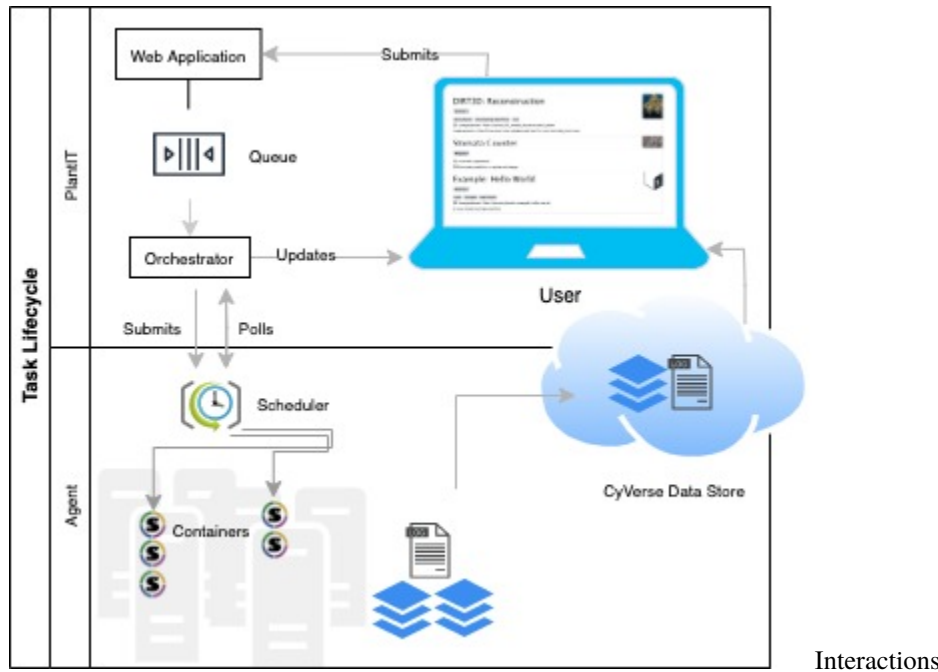
An **Agent** is a deployment target: an abstraction of a cluster or supercomputer along with SLURM scheduler configuration details.

2.2.3 Workflows

A **Workflow** is an executable research application packaged into a [Docker](#) image. Workflows are deployed in the [Singularity](#) container runtime. To define a workflow, add a `plantit.yaml` file to any public [GitHub repository](#).

2.2.4 Tasks

A **Task** is an instance of a workflow, deployed to an agent. When a task is submitted from the browser, the plantit web app hands it to an internal queue feeding an orchestrator process. When the orchestrator picks up the task, it generates a job script and submits it to the selected cluster/supercomputer scheduler, then monitors its progress until completion.



2.2.5 Projects

A **Project** is a MIAPPE investigation, which may contain one or more studies. MIAPPE (Minimum Information About a Plant Phenotyping Experiment) is a formal ontology for organizing data, metadata, experiments, and analyses. plantit allows datasets and tasks to be freely associated with MIAPPE projects.

2.3 Submitting tasks

2.3.1 Select a workflow

To explore workflows, navigate to the **Workflows** tab from the home view.

The screenshot shows the 'plantit v0.0.2-alpha' web interface. The top navigation bar includes links for About, Stats, Status, Docs, and Github. A sidebar on the left contains links for Workflows, Datasets, Projects, Agents, Tasks, and Users. The main content area is titled 'Workflows' and is powered by GitHub. It displays a grid of workflow cards under the 'Featured' context. The cards are:

- DIRT2D** by Alexander Bucksch: Tags include python3, computer-vision, image-processing, imaging, phenotyping, phenotyping-algorithms, and root. Description: Digital Imaging of Root Traits: Extract trait measurements from images of monocot and dicot roots.
- DIRT3D Reconstruction** by Suxing Liu: Tags include master, phenotyping, phenotyping-algorithms, and root. Description: Implementation of the 3D reconstruction pipeline optimized for plant branching structures.
- SMART: Speedy Measurement of Arabidopsis Rosette Traits** by Suxing Liu: Tags include master, computer-vision, extract-traits, image-segmentation, and phenotyping. Description: Extract geometric traits from top-view images of plants.
- Stomata Counter** by Karl Fetter, Sven Eberhardt: Tags include master, w-bonelli/epidermal. Description: DCN stomata prediction on epidermal images.
- DIRTmu** by Peter Pietrzyk: Tags include main. Description: Automatic root hair extraction from microscopy images.

Workflows

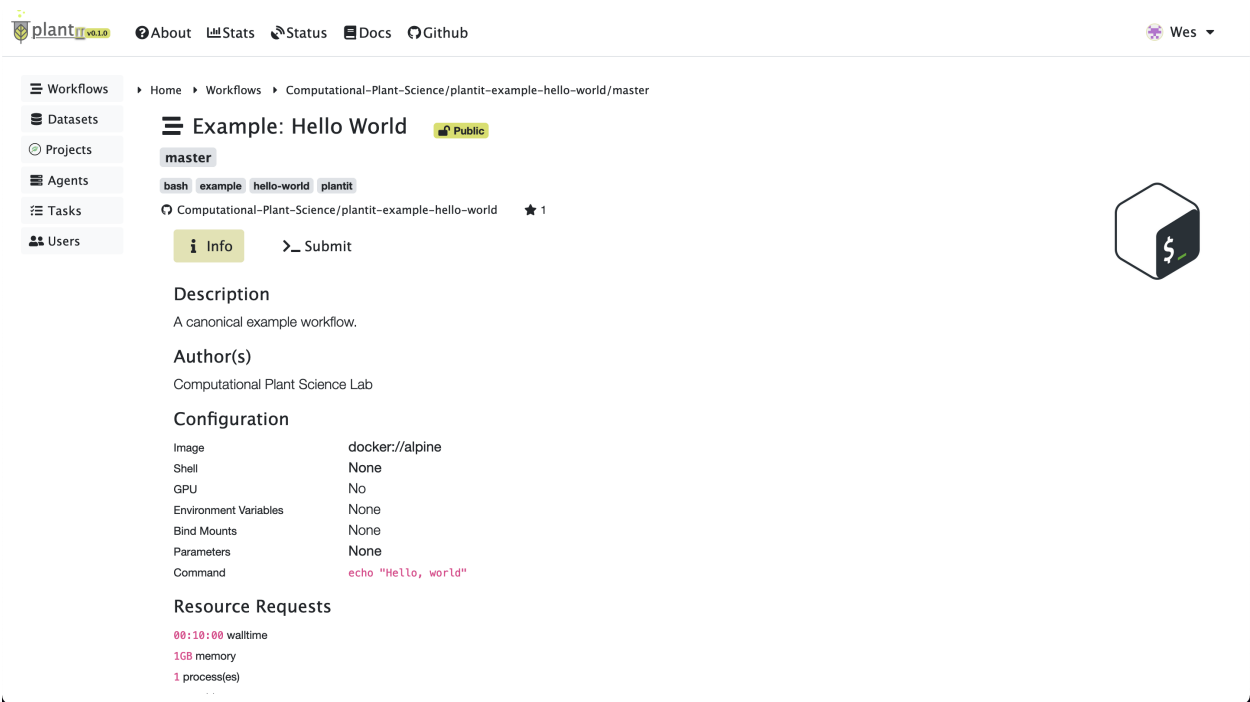
By default, this page will display the **Featured** workflow context: a curated set of applications provided by the Computational Plant Science lab, collaborators, and other researchers.

Click the **Featured** dropdown to select a different context. Options include:

- **Examples:** a small set of simple workflows to serve as templates and examples
- **Public:** all publicly available workflows
- **Yours:** your own workflows (private and public)
- **[Organization]:** workflows belonging to a particular organization
- **[Project]:** workflows associated with a particular MIAPPE project

Select a workflow to view its authorship, related publications, parameter list, and deployment configuration.

2.3.2 Submit to an agent



information

To configure and submit a task for the workflow you’ve selected, click **Submit**. This will present some configuration options including (at least):

- ID: the task’s (unique) identifier
- Tags: arbitrary text tags to associate with the task
- Time: the task’s time limit
- Agent: the agent to submit the task to
- Output: the folder to deposit results in

If the workflow requires input files or parameters, corresponding configuration sections will be shown.

plantit v0.0.2-alpha About Stats Status Docs Github

Wes

Workflows Home Workflows Computational-Plant-Science/plantit-example-hello-world/master

Datasets

Projects

Agents

Tasks

Users

Example: Hello World

Public

master

bash example hello-world plantit

Computational-Plant-Science/plantit-example-hello-world ★ 1

Info Submit

Last run 43 minutes ago (March 30th 2022, 10:18 pm)

Sections marked with ! are required.

+ # ID 5b86c5e8-47b3-4666-8b1e-609fda936092 ✓

+ ⌚ Time 1 Hours ✓

- 📁 Output /iplant/home/wbonelli/analyses ✓
Select a directory in the CyVerse Data Store to transfer results to.
wbonelli

+ 🧑 Agent Stampede2 ✓

submission

After all fields have been configured, click the **Start** button to submit the task.

2.3.3 Monitor status

After a moment the task page will appear. At first there may be no log messages.

plantit v0.0.2-alpha About Stats Status Docs Github

Wes

Workflows Home Tasks wbonelli/73d37c92-864a-4724-8185-b49f3d04b54a

Datasets

Projects

Agents

Tasks

Users

73d37c92-864a-4724-8185-b49f3d04b54a

CREATED on Stampede2

GUID Cancel

Example: Hello World

Computational Plant Science Lab

master

bash example hello-world plantit

Computational-Plant-Science/plantit-example-hello-world

A canonical example workflow.

Created a few seconds ago Last updated a few seconds ago Due in an hour

22:18:13.999 Mar 30, 2022 22:18:13.9995 22:18:14 22:18:14.0005 22:18:14.001

Task status: **CREATED**

Before long the task should be created, scheduled, and started on the appropriate agent. At this point you should see a few lines of log output:

The screenshot shows the plantit v0.0.2-alpha web interface. The top navigation bar includes links for About, Stats, Status, Docs, and Github. The user is logged in as 'Wes'. The left sidebar shows a menu with Workflows, Datasets, Projects, Agents, Tasks, and Users. The main content area displays a task titled '73d37c92-864a-4724-8185-b49f3d04b54a' which is 'RUNNING on Stampede2'. The task is located at '/iplant/home/wbonelli/analyses'. Below the task title, there is a section titled 'Example: Hello World' with a 'master' branch and a 'bash example hello-world plantit' command. The task was created a few seconds ago, last updated a few seconds ago, and is due in an hour. The log output shows the task creation and preparation of the environment.

Task

status: RUNNING

When a task completes successfully, the status will change from RUNNING to COMPLETED.


The screenshot shows the plantit v0.0.2-alpha web interface with the same task '73d37c92-864a-4724-8185-b49f3d04b54a' now in a 'COMPLETED on Stampede2' state. The task is still located at '/iplant/home/wbonelli/analyses'. The 'Example: Hello World' section remains the same. The task was created 40 minutes ago, ran for 7 minutes, and completed 33 minutes ago. The log output shows the task completion and the results found.


Task


status: COMPLETED

2.3.4 Retrieve results

The output folder in the CyVerse data store section will eventually open at the bottom of the view (you may need to reload the page). Results will be zipped into a file with name matching the task’s ID.

 [About](#) [Stats](#) [Status](#) [Docs](#) [Github](#)

 Wes ▾

 CYVERSE Data Store

wbonelli41 folders, 0 files+↺▾

agdrone_cc_results▴

analyses0 folders, 3 filesShared with 1 user(s)Bind project/study+🗑️↺🔗▾

73d37c92-864a-4724-8185-b49f3d04b54a.zip1.862 KB👁️🗑️👤

b0acaebf-3cd2-477b-ad6e-3962b3fefe33.zip3.569 KB👁️🗑️👤

e5a92bb0-0a5d-4039-8193-fd5b7a4f673d.zip1.689 KB👁️🗑️👤

Task

results

10

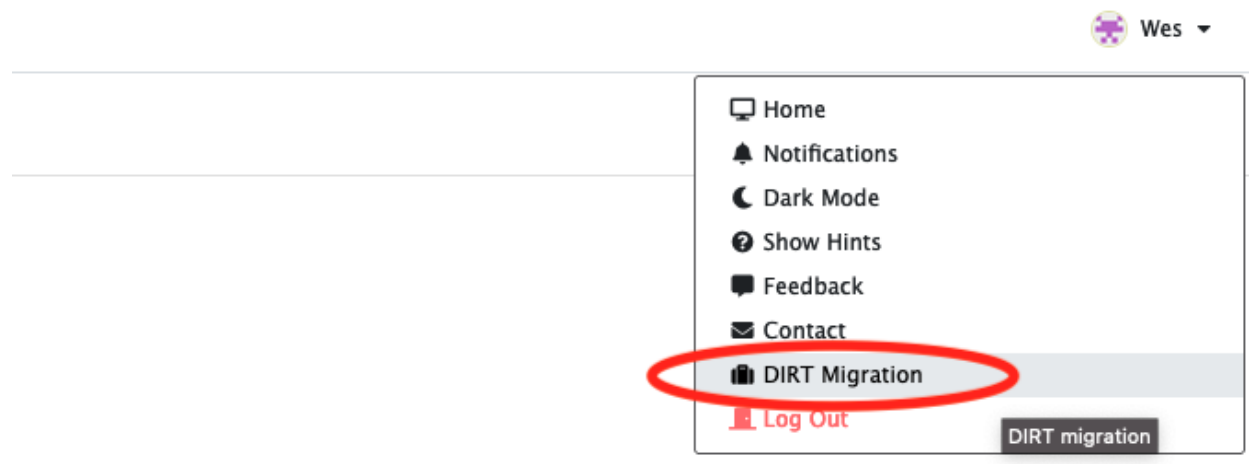
Chapter 2. Quickstart

DIRT MIGRATION

- *Migrating DIRT datasets*

3.1 Migrating DIRT datasets

DIRT users may elect to migrate their datasets stored in the DIRT system to the CyVerse data store for use with `plantit`. This process can be started by selected the drop-down profile menu in the top-right of the `plantit` web UI:



DIRT migration

`plantit` will transfer each of your DIRT datasets to a correspondingly named subcollection within a newly created collection in the CyVerse data store. This collection will have path `/iplant/home/{your username}/dirt_migration`. (If for some reason you already have a collection with the same name, you will be prompted to rename it before the migration can proceed.) Once you have started the migration, progress and completion status will be shown onscreen.

WORKFLOWS

- *Using workflows*
- *Binding workflows: the `plantit.yaml` file*
 - *Required attributes*
 - * *Name*
 - * *Author*
 - * *Image*
 - * *Commands*
 - *Optional attributes & sections*
 - * *Public*
 - * *Email*
 - * *Shell*
 - * *GPUs*
 - * *Environment variables*
 - * *Parameters*
 - *Default values*
 - * *Input/output*
 - *Inputs*
 - *Input types (file, files, and directory)*
 - *Input filetypes*
 - *Outputs*
 - * *Jobqueue configuration*
 - *Walltime*
 - *Virtual memory*
- *A simple example*
- *Repository refresh rate*

A **Workflow** is `plantit`'s unit of work. Workflows can be research applications of nearly any kind: image processing, growth simulations, even genome analysis, although we encourage you to first consider whether other free and open source bioinformatics tools ([CoGe](#), [easyGWAS](#)) or commercial services ([Benchling](#), [Latch](#)) meet your needs.

4.1 Using workflows

For a tutorial on exploring and submitting plantit workflows, see the [quickstart](#).

4.2 Binding workflows: the plantit.yaml file

To host a **Workflow** on plantit, add a plantit.yaml file to some branch in any public GitHub repository.

4.2.1 Required attributes

At minimum, the plantit.yaml file should look something like this:

```
name: Hello World
author: Groot
image: docker://ubuntu
commands: echo "I am Groot!"
```

There are five required attributes:

- **name**: the name of the workflow
- **author**: the workflow's author(s)
- **image**: the Docker image to use
- **commands**: the workflow's entry point

Name

The name the workflow will appear under in the plantit web UI. Need not be unique, however it's generally best to try to give each workflow a unique, distinctive, relevant name. (This will help potential users of your software to find it via keyword search.)

Author

The workflow's author(s) or developer(s). Must be a string, although future versions of plantit may support a list of strings.

Image

The Docker image to run the workflow with. Must be publicly available on [Docker Hub](#). It's a good idea to read the Singularity [documentation on Docker/OCI interop](#) before binding a workflow to plantit. Most phenotyping software is unlikely to encounter issues, but there are a few Singularity runtime compatibility caveats to be aware of.

Commands

Your workflow's entry point. If you have multiple things to do, it's generally best to put them in a script, rather than using `&&` to append commands. (This may work for some image definitions, but is not guaranteed to.)

4.2.2 Optional attributes & sections

There are a number of optional properties and sections as well:

- **public**: whether the workflow should be accessible to all `plantit` users (defaults to `false`)
- **email**: your (the workflow developer's) preferred email address, e.g. for support/questions
- **shell**: the shell to use to invoke your entry point (`sh`, `bash`, or `zsh`, defaults to `bash`)
- **gpu**: whether this workflow should use GPUs (if available)
- **env**: environment variables to provide to your container runtime(s)
- **params**: parameters to be configured at submission time in the `plantit` web UI
- **input**: the kind, names, patterns, and optionally, the default path of any inputs to the workflow
- **output**: the kind, names and patterns of results expected from the workflow
- **jobqueue**: the resources to request from the cluster scheduler

Public

By default workflows are only visible to you in the `plantit` web UI. To make a workflow publicly accessible to all users, set the `public` attribute to `true`.

Email

You can provide a contact email via an `email` attribute. If this attribute is provided, a `mailto` link will be shown in the user interface to allow your workflow's users to easily contact you.

Shell

By default, the `command` specified in `plantit.yaml` is invoked directly from the Singularity container runtime, i.e., `singularity exec <image> <command>`. Since [Singularity runs in a modified shell environment](#) some behavior may differ from that produced by Docker. Some Anaconda-based images can be configured to automatically activate an environment, for instance. With Singularity this cannot be achieved without wrapping the command with `bash -c '<command>'` and [editing bash startup files in the container definition](#).

For these reasons `plantit` provides a `shell` option. If provided, this option will cause `plantit` to wrap the `command` with `... <shell> -c 'command'` when invoking Singularity. Supported values include:

- `bash`
- `sh`
- `zsh`

GPUs

To indicate that your workflow can take advantage of GPUs (only available on select deployment targets) add a `gpu: True` line to your configuration file. When deployed to environments with GPUs, your task will have access to an environment variable `$GPUS`, set to the number of GPU devices provided by the host.

Environment variables

Certain environment variables will be automatically set in the Singularity container runtime when a workflow is submitted as a task, in case you need to reference them in your entry point command or script:

- `$WORKDIR`: the current working directory
- `$INPUT`: the path to an input file or directory
- `$OUTPUT`: the path to the directory results will be written to
- `$INDEX`: the index of the current input file (if there are multiple, otherwise defaults to 1 for single-file or -directory tasks)

You can provide custom environment variables in an `env` section, for instance:

```
...
env:
- LC_ALL=C.UTF-8
- LANG=C.UTF-8
...
```

Parameters

To parametrize your workflow, add a `params` section. For example, to allow the user to configure the message printed by the trivial workflow above:

```
name: Hello Who?
author: Groot
public: True
clone: False
image: docker://alpine
commands: echo "$MESSAGE"
params:
- name: message
  type: string
```

This will cause the value of `message`, specified in the `plantit` web UI at task submission time, to be substituted for `$MESSAGE` in the `command` at runtime.

Four parameter types are supported by `plantit`:

- `string`
- `select`
- `number`
- `boolean`

See the `Computational-Plant-Science/plantit-example-parameters` workflow [on GitHub](#) for an example of how to use parameters.

Default values

To provide default values for your workflow's parameters, you can use a `default` attribute. For instance:

```
params:
- name: message
  type: string
  default: 'Hello, world!'
```

Input/output

plantit can automatically copy input files from the [CyVerse Data Store](#) or [Data Commons](#) onto the file system in your deployment environment, then push results back to the Data Store after your task completes. To configure inputs and outputs for a workflow, add `input` and `output` attributes to your configuration.

Inputs

If your workflow requires inputs, add an `input` section to your configuration file, containing at minimum a `path` attribute (pointing either to a directory- or file-path in the CyVerse Data Store or Data Commons, or left blank) and a `kind` attribute indicating whether this workflow operates on a single file, multiple files, or an entire directory.

Input types (file, files, and directory)

To indicate that your workflow should pull a single file from the Data Store and spawn a single container to process it, use `kind: file`. To pull a directory from the Data Store and spawn a container for each file, use `kind: files`. To pull a directory and spawn a single container to process it, use `kind: directory`.

It's generally a good idea for `path` to reference a community-released or curated public dataset in the CyVerse Data Commons, so prospective users can test your workflow on real data. For instance, the `plantit.yaml` for a workflow which operates on a single file might have the following `input` section

```
input:
  path: /iplant/home/shared/iplantcollaborative/testing_tools/cowsay/cowsay.txt
  kind: file
```

Input filetypes

To specify which filetypes your workflow is permitted to accept, add a `filetypes` attribute to the `input` section:

```
input:
  path: /iplant/home/shared/iplantcollaborative/testing_tools/cowsay
  kind: file
  filetypes:
    - txt
```

Any values provided to `filetypes` will be joined (with `,`) and substituted for `FILETYPES` in your workflow's command. Use this to inform your code which filetypes to expect, for example:

```
commands: ls "$INPUT"/*.{"FILETYPES"} >> things_cow_say.txt
input:
  path: /iplant/home/shared/iplantcollaborative/testing_tools/cowsay
  kind: file
  filetypes:
    - txt
    - md
```

Note that while the `input.path` and `input.filetypes` attributes are optional, you must provide a `kind` attribute if you provide an `input` section.

Outputs

If your workflow produces outputs, add an `output` section with a `path` attribute to your configuration file. This attribute may be left blank if your workflow writes output files to the working directory; otherwise the value should be a directory path relative to the working directory. For example, to indicate that your workflow will deposit output files in a directory `output/directory` relative to the workflow's working directory:

```
output:
  path: output/directory
```

By default, all files under the given path are uploaded to the location in the CyVerse Data Store provided by the user. To explicitly indicate which files to include/exclude (this is suggested especially if you workflow deposits files in the working directory), add `include` and `exclude` sections under `output`:

```
output:
  path: output/directory
  include:
    patterns:           # include excel files
    - xlsx              # and png files
    names:
    - important.jpg     # but only this .jpg file
  exclude:
    patterns:
    - temp              # don't include anything marked temp
    names:
    - not_important.xlsx # and exclude a particularexcel file
```

If only an `include` section is provided, only the file patterns and names specified will be included. If only an `exclude` section is present, all files except the patterns and names specified will be included. If you provide both `include` and `exclude` sections, the `include` rules will first be applied to generate a subset of files, which will then be filtered by the `exclude` rules.

Jobqueue configuration

To ensure your workflow takes optimal advantage of cluster resources, add a `jobqueue` section. To indicate that instances of your workflow should request 1 process and 1 core on 1 node with 1 GB of memory with 1 hour of walltime:

```
jobqueue:
  walltime: "01:00:00"
  memory: "1GB"
  processes: 1
  cores: 1
```

If a `jobqueue` section is provided, all four attributes are required. If you do not provide a `jobqueue` section, tasks will request 1 hour of walltime, 10 GB of RAM, 1 process, and 1 core on all agents.

Walltime

When a `plantit` task is submitted, values provided for the `jobqueue.walltime` attribute will be passed through transparently to the selected deployment target's scheduler. The `plantit` web UI will timestamp each task submission. If such a time limit is provided at submission time, `plantit` will attempt to cancel your task if it fails to complete before the time limit has elapsed.

Virtual memory

Note that some deployment targets (namely the default public agent, [TACC's Stampede2](#)) are equipped with virtual memory. For tasks deployed to agents with virtual memory, `plantit` will ignore values provided for the `jobqueue.memory` attribute and defer to the cluster scheduler: on Stampede2, for instance, all tasks have access to 98GB of RAM.

4.3 A simple example

The following workflow prints the content of an input file and then writes it to an output file located in the same working directory.

```
name: Hello File
image: docker://alpine
public: True
commands: cat "$INPUT" && cat "$INPUT" >> cowsaid.txt
input:
  from: /iplant/home/shared/iplantcollaborative/testing_tools/cowsay/cowsay.txt
  kind: file
output:
  path: # blank to indicate working directory
  include:
    names:
      - cowsaid.txt
```

4.4 Repository refresh rate

The `plantit` web application scrapes GitHub for repository information for all logged-in users every 5 minutes. (If you've just updated a repository, you may need to wait several minutes then reload the workflow page.)

DATASETS

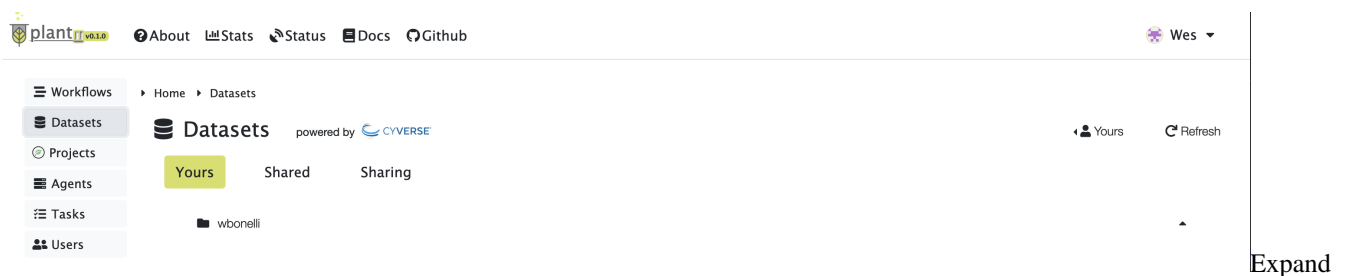
- *Viewing data*
- *Downloading data*
- *Sharing data*
- *Deleting data*

A **Dataset** is a folder in the CyVerse data store; whether in the user's personal directory, or in the community iplant/home/shared/ folder, or in the public data commons.

5.1 Viewing data

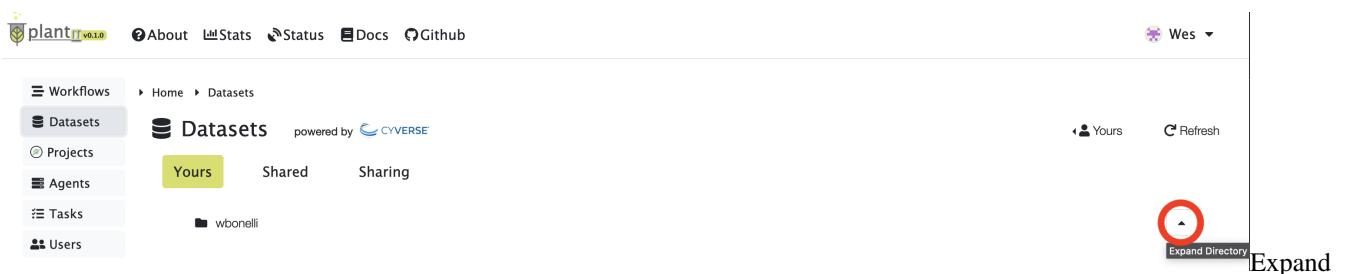
To view datasets, navigate to the **Datasets** tab from the home view. This will present a page with three (3) tabs:

- **Yours:** your own personal datasets
- **Shared:** datasets other users have shared with you
- **Sharing:** datasets you're sharing with other users



folder

To expand a folder in the data tree, click the caret button on the right side of the frame.



folder

5.2 Downloading data

To download a file from the CyVerse data store, click the download button on the right side of the menu.

The screenshot shows the plantit v0.0.2-alpha interface. The top navigation bar includes links for About, Stats, Status, Docs, and Github. The main content area is titled 'Datasets' and is powered by CYVERSE. The 'Yours' tab is selected, showing a list of datasets under the user 'wbonelli'. The 'agdrone_cc_results' folder is expanded, showing a list of 7 files. The 'Download File' button is highlighted with a red circle.

File Name	Size	Actions
19675bd2-b4c6-4d5a-954a-5bada2c426e3.zip	4.406 KB	Download File
1dab651f-cb6e-47c9-8c0e-32b6a694457e.zip	5.776 KB	Download File
50d59bfc-ccaf-40d9-ab67-c8df03f9373c.zip	5.777 KB	Download File
88e42cb2-7df1-4a6c-acc0-19d1ec2ea765.zip	4.186 KB	Download File
8f5de2c6-b6fe-4713-a4c3-6ed02478e378.zip	1.805 KB	Download File
9cfead40-cced-4185-bc50-f4d24638be37.zip	4.747 KB	Download File
e4c1e6a3-6899-4cfe-9b33-4b3cf501650c.zip	5.747 KB	Download File

Download

file

The data tree will disable itself while the file downloads.

The screenshot shows the plantit v0.0.2-alpha interface. The top navigation bar includes links for About, Stats, Status, Docs, and Github. The main content area is titled 'Datasets' and is powered by CYVERSE. The 'Yours' tab is selected, showing a list of datasets under the user 'wbonelli'. The 'agdrone_cc_results' folder is expanded, showing a list of 7 files. The 'Download File' button is highlighted with a red circle. A loading spinner is visible in the center of the file list.

File Name	Size	Actions
19675bd2-b4c6-4d5a-954a-5bada2c426e3.zip	4.406 KB	Download File
1dab651f-cb6e-47c9-8c0e-32b6a694457e.zip	5.776 KB	Download File
50d59bfc-ccaf-40d9-ab67-c8df03f9373c.zip	5.777 KB	Download File
88e42cb2-7df1-4a6c-acc0-19d1ec2ea765.zip	4.186 KB	Download File
8f5de2c6-b6fe-4713-a4c3-6ed02478e378.zip	1.805 KB	Download File
9cfead40-cced-4185-bc50-f4d24638be37.zip	4.747 KB	Download File
e4c1e6a3-6899-4cfe-9b33-4b3cf501650c.zip	5.747 KB	Download File

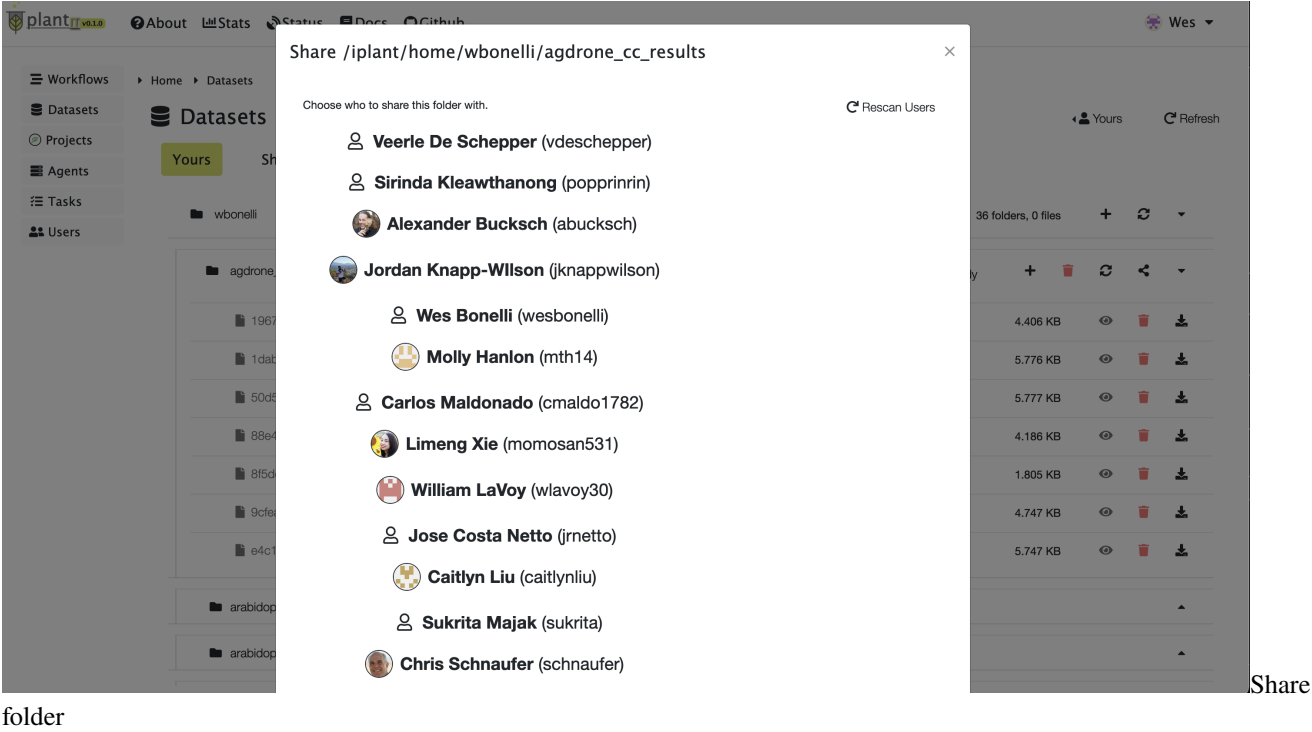
Download

file

Finally a download popup will appear.

5.3 Sharing data

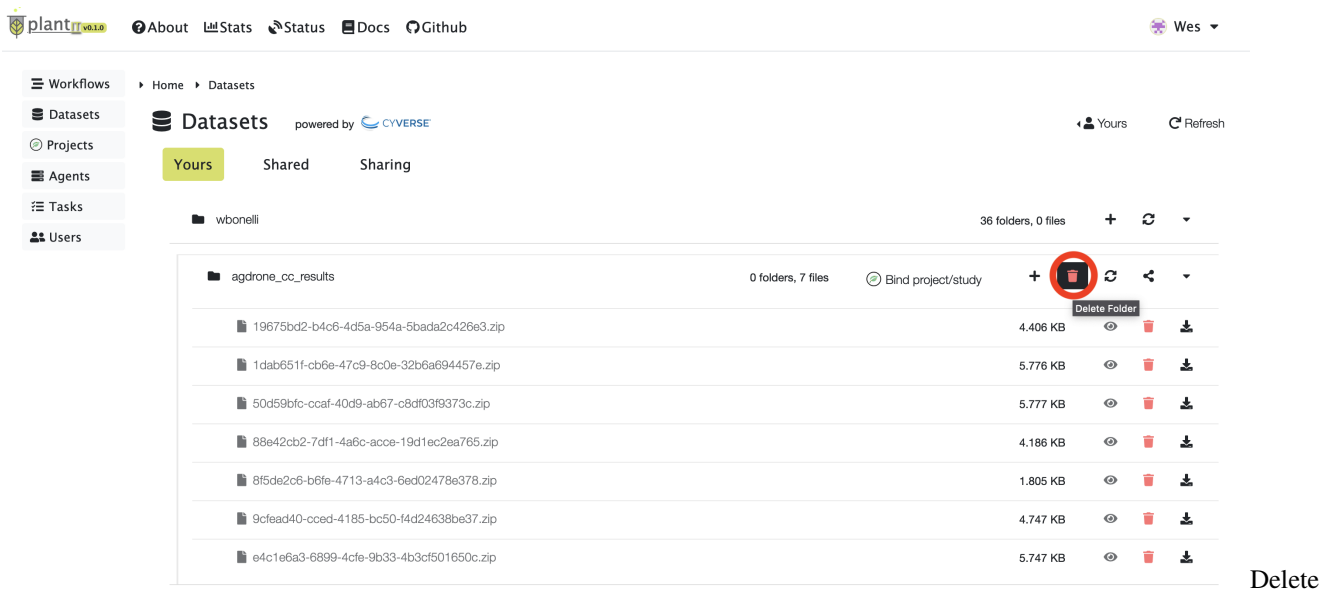
To share a folder with another user, select the share button on the right side of the menu, then select one or more users to share the folder with.



After sharing a folder, the receiving user will be able to access it under their Shared datasets tab.

5.4 Deleting data

To delete a personal file or folder in the data store, select the red delete button on the right side of the menu.



data

AGENTS

- *Public agents*
- *Integrating a new agent*

An **Agent** runs **Tasks**. Abstractly, an agent is a particular way of connecting and submitting to a cluster, supercomputer or server equipped with the SLURM scheduling system. Multiple agents can be configured for the same underlying systems, e.g. to permit submissions to distinct queues.

6.1 Public agents

By default `plantit` provides a single public agent to all users, free of charge, called `Stampede2`.

The screenshot shows the `plantit` web interface. The top navigation bar includes links for About, Stats, Status, Docs, and Github, along with a user profile icon labeled 'Wes'. The left sidebar contains a menu with options: Workflows, Datasets, Projects, Agents (selected), Tasks, and Users. The main content area displays the configuration for the 'Stampede2' agent, which is associated with the University of Texas Advanced Computing Center's Stampede2 cluster. The configuration details include:

- Configuration:** scheduler is 'slurm', working directory is '/scratch/03203/dirt/plantit', and setup commands are 'export LC_ALL=en_US.utf8 && export LANG=en_US.utf8 && ml load python3 && ml load tacc-singularity/3.7.2 && ml load launcher/3.9'.
- Resources (per container):** 12 core(s), 12 process(es), Virtual memory, and No GPU.
- Connection Status:** A timeline from 05:00 to 11:00 on Apr 1, 2022, showing green bars indicating active periods.

 A 'Check' button is visible in the top right corner of the configuration panel. The interface is labeled 'Stampede2' in the bottom right corner.

This agent submits jobs to the Stampede2 cluster at the Texas Advanced Computing Center at the University of Texas. Stampede2 nodes have (up to 96GB of) virtual memory and will ignore workflow-specific memory requests, instead automatically allocating memory as needed. Batch jobs are submitted in parallel where possible to accelerate processing. Completion times may vary widely depending on the cluster's availability and workload at any given moment.

6.2 Integrating a new agent

Please [contact the plantit developers](#) if you would like to bind a cluster or other deployment target at your institution. User-managed deployment targets must be available via key-accessible SSH — administrators will be provided a public key to add to their system’s SSH server configuration, allowing plantit remote access. Clusters may be maintained for private use or made available to all plantit users. A future version of plantit may support agent management directly in the web UI.

TASKS

- *Task monitoring*
- *Task lifecycle*

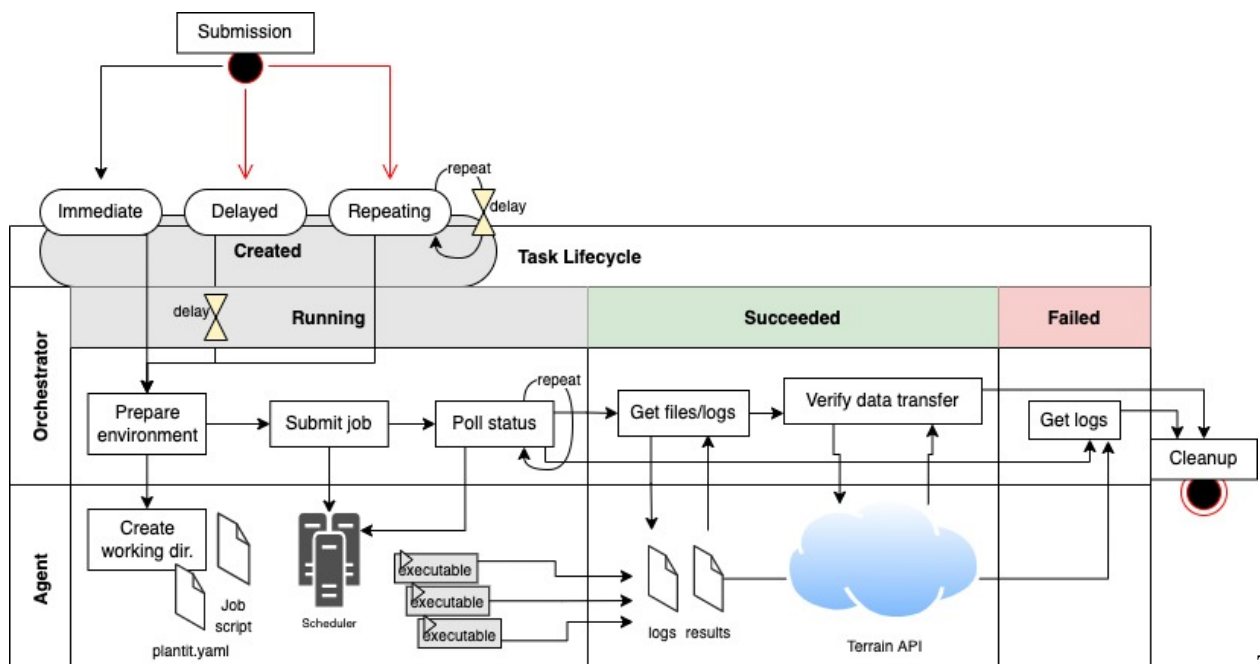
A **Task** is a single instance of a workflow. When a task is submitted from the browser, the `plantit` web app hands it to an internal queue feeding a background worker. When the worker picks up the task, a job script is generated and submitted to the selected cluster/supercomputer scheduler. The task lifecycle is a simple state machine strung together from Celery tasks.

7.1 Task monitoring

For a tutorial on monitoring a task and retrieving results, see the [quickstart](#).

7.2 Task lifecycle

The task lifecycle is a state machine progressing from `CREATED` to `RUNNING` to one of several mutually exclusive final states (`COMPLETED`, `FAILED`, `TIMEOUT`, or `CANCELLED`).



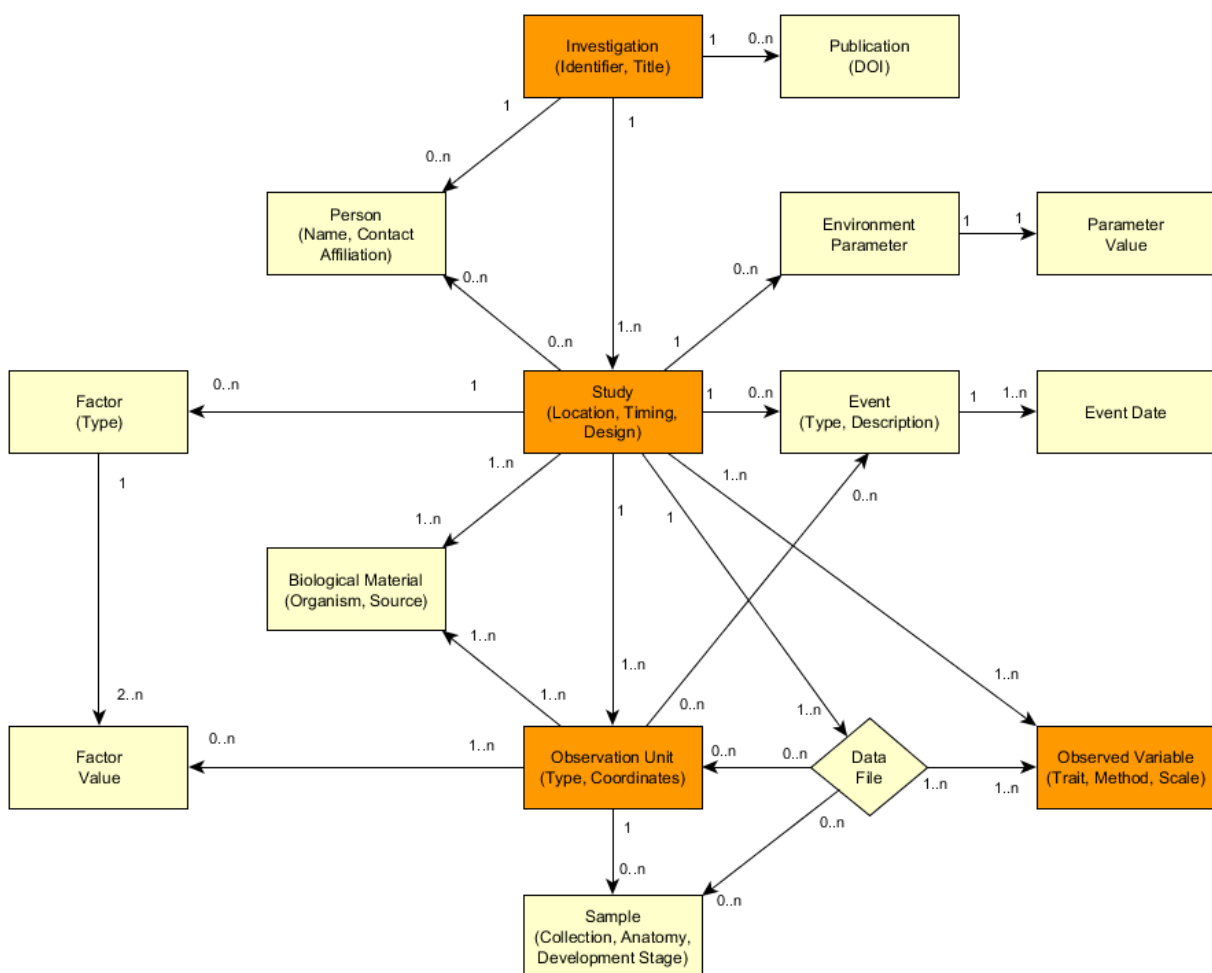
Task

Lifecycle

When a task successfully completes, results are automatically transferred to the selected location in the CyVerse data store. The user is then shown results produced and may download them from the browser individually or bundled into a single archive.

PROJECTS

A **Project** is a **MIAPPE** investigation: a formal ontology for metadata describing a phenotyping experiment.



Entity

relational diagram

The MIAPPE schema permits one or more studies to be associated with each investigation. Each study describes a particular instance of an experiment, with properties such as start/end dates, location, environmental parameters, etc.

Currently **plantit** allows users to associate projects and studies with tasks, linking computational analyses to their experimental context. Eventually we also intend to support:

- associating studies with one or more datasets

- tagging data objects as samples of biological materials
- image annotations

EXAMPLES

- *Hello world*
- *Parameters*
- *Accessing data*

Several example workflows are provided as templates. They demonstrate essential features, namely:

- invoking a command
- using parameters
- accessing data

9.1 Hello world

This is just about the simplest possible workflow it's possible to host on `plantit`.

TODO: describe log files in results

9.2 Parameters

This `workflow` demonstrates the use of user-configurable workflow parameters.

9.3 Accessing data

This `workflow` shows how to construct a workflow with inputs and outputs.

ARCHITECTURE

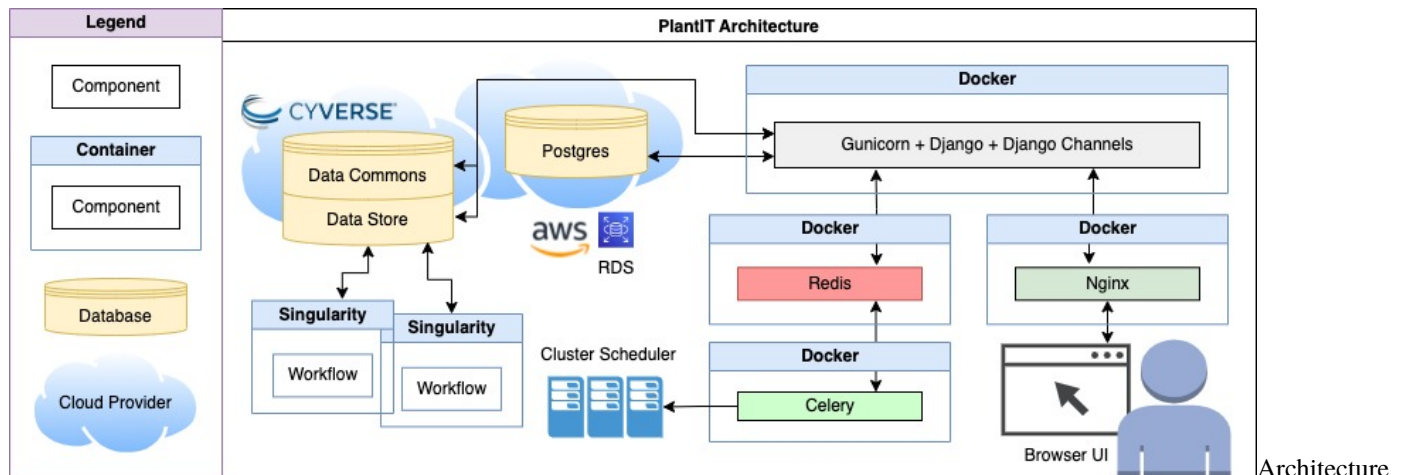
- *Motivation*
- *Technologies*

10.1 Motivation

`plantit` is middleware binding GitHub and CyVerse to various institutional clusters & supercomputers via a web interface. Broadly, `plantit` aims to reinvent as little as possible, gluing existing tools together in ways that make data-intensive plant phenomics accessible and reproducible.

10.2 Technologies

The `plantit` stack is predominantly Python. The backend is Nginx, Gunicorn, Django + Channels, Celery, Redis, & Postgres, orchestrated with Docker Compose. The frontend is Vue.



Feel free to [reach out](#) if you'd like to contribute to `plantit` development, start your own instance of `plantit` somewhere, modify it for another discipline, etc.

CONTRIBUTIONS

- *Configuring a development environment*
- *Command cheatsheet*
 - *Docker Compose*
 - *Docker*
 - *Django*

We welcome contributions to the `plantit` codebase, from [bug reports](#) to documentation fixes to pull requests of all kinds! All development planning is carried out on GitHub: see the [Changelog/Roadmap](#) and [Issues](#) in particular.

11.1 Configuring a development environment

See the [README](#) for instructions on installing the project from source.

11.2 Command cheatsheet

Below is a list of handy commands for managing the `plantit` application.

11.2.1 Docker Compose

- `docker-compose -f docker-compose.dev.yml up`: bring the full application (all containers) up
- `docker-compose -f docker-compose.dev.yml down`: bring the full application (all containers) down
- `docker-compose -f docker-compose.dev.yml run plantit <command>` run a command in the `plantit` container (starting containers as needed)
- `docker-compose -f docker-compose.dev.yml exec plantit <command>` run a command in the `plantit` container (assumes all containers are up)

11.2.2 Docker

- `docker ps`: list running containers
- `docker exec -it <container ID> bash`: enter an already running container

11.2.3 Django

- `./manage.py`: list Django commands
- `./manage.py makemigrations`: create plan for django migrations
- `./manage.py migrate`: run django migrations
- `./manage.py shell`: opens Django Python interpreter