Terminus

Release 0.7

Damien CHAPON ; Loic STRAFELLA

Aug 09, 2022

DOCUMENTATION

CHAPTER

ONE

INTRODUCTION

The purpose of the `galactica-terminus` Python package is to provide computational astrophysicists a way to give access to their numerical simulation (raw) data via custom post-processing services made available to the community through the Galactica simulation database web application.

This package lets you configure a `Terminus` deamon on a machine with computational resources you want to allocate to the dedicated task of simulation data post-processing (data extraction, analysis, visualization, etc.). It is based on the Celery Python asynchronous task management framework, using RabbitMQ as a message broker to communicate with the Galactica simulation database.

1.1 Installation

1.1.1 Latest stable releases

Using **pip**, you can easily download and install the latest version of the `galactica-terminus` package directly from the Python Package Index (PyPI):

```
> pip install galactica-terminus
  Collecting celery>=5.0.1
    Downloading celery-5.0.1-py3-none-any.whl (392 kB)
    || 392 kB 18.2 MB/s
  Collecting click-repl>=0.1.6
    Using cached click_repl-0.1.6-py3-none-any.whl (4.2 kB)
  Collecting click>=7.0
    Using cached click-7.1.2-py2.py3-none-any.whl (82 kB)
  Processing ./.cache/pip/wheels/03/a8/59/
↪811a351d002c7fca47dc94c622f36a9bf6e27fa9da49c8ec0c/click_didyoumean-0.0.3-py3-none-any.
↪whl
  Collecting billiard<4.0,>=3.6.3.0
    Using cached billiard-3.6.3.0-py3-none-any.whl (89 kB)
  Collecting vine<6.0,>=5.0.0
    Using cached vine-5.0.0-py2.py3-none-any.whl (9.4 kB)
  Collecting kombu<6.0,>=5.0.0
    Using cached kombu-5.0.2-py2.py3-none-any.whl (180 kB)
```

(continues on next page)

(continued from previous page)

```
  Collecting pytz>dev
    Using cached pytz-2020.1-py2.py3-none-any.whl (510 kB)
  Collecting prompt-toolkit
    Using cached prompt_toolkit-3.0.8-py3-none-any.whl (355 kB)
  Collecting six
    Using cached six-1.15.0-py2.py3-none-any.whl (10 kB)
  Collecting importlib-metadata>=0.18; python_version < "3.8"
    Using cached importlib_metadata-2.0.0-py2.py3-none-any.whl (31 kB)
  Collecting amqp<6.0.0,>=5.0.0
    Using cached amqp-5.0.1-py2.py3-none-any.whl (46 kB)
  Collecting wcwidth
    Using cached wcwidth-0.2.5-py2.py3-none-any.whl (30 kB)
  Collecting zipp>=0.5
    Downloading zipp-3.4.0-py3-none-any.whl (5.2 kB)
  Installing collected packages: click, wcwidth, prompt-toolkit, six, click-repl, click-
↪didyoumean, billiard, vine, zipp, importlib-metadata, amqp, kombu, pytz, celery, gal-
↪terminus
  Successfully installed amqp-5.0.1 billiard-3.6.3.0 celery-5.0.1 click-7.1.2 click-
↪didyoumean-0.0.3 click-repl-0.1.6 gal-terminus-0.5rc1 importlib-metadata-2.0.0 kombu-5.
↪0.2
                         prompt-toolkit-3.0.8 pytz-2020.1 six-1.15.0 vine-5.0.0 wcwidth-
↪0.2.5 zipp-3.4.0
```

1.2 Galactica access setup

In order to communicate with the Galactica simulation database, your `Terminus` instance interacts with a messaging server (based on the AMQP protocol) called RabbitMQ. This messaging system allows the `Terminus` server to :

- receive **job request messages** from the Galactica web app., containing information on :
  - which post-processing service to execute,
  - which simulation raw data to post-process,
  - what are the post-processing service parameter values requested by the user.
- report back **job status update messages** to the Galactica web app., in order to :
  - display job status message on the user's personal *job requests* page,
  - send the user email notifications if his/her *job request* failed or succeeded.

Upon successful job completion, the produced data tarball needs to be uploaded (using a basic **`rsync`** command) to the Galactica server so that the user who submitted the job request can download it from the web application.

To configure these two ways of communication between your `Terminus` instance and the Galactica web app.:

- **RabbitMQ messages** : see *Get RabbitMQ credentials*,
- **rsync** : see *SSH configuration*.

1.2.1 Get RabbitMQ credentials

To get RabbitMQ credentials, please kindly send an email to the Galactica database administrator (admin@galactica-simulations.eu), providing the following information :

- *Research institute* hosting the server on which you want to deploy Terminus,
- Server `hostname` or a custom alias of your choice (see also `terminus_host_name` in *Terminus configuration*),
- Server `IP address`.

The Galactica database administrator will provide you the necessary parameters to setup `Terminus` (see *Terminus configuration*) :

- a RabbitMQ user name
- a RabbitMQ password
- a RabbitMQ host
- a RabbitMQ port
- a RabbitMQ virtual host
- a SSH target configuration (to add into your `~/.ssh/config`).

Note: You do not necessarily have to create an account on the Galactica simulation database to configure and run a `Terminus` server linked to the database. You only require RabbitMQ credentials and (obviously) system administration permissions on the machine.

1.2.2 SSH configuration

Providing a SSH public key

So `Terminus` can upload result tarballs from successful jobs, the user running the `Terminus` server instance need to create a (passphrase-less) SSH key pair:

```
$ ssh-keygen -t ed25519 -f ~/.ssh/id_ed25519_gal-terminus
    Generating public/private rsa key pair.
    Enter passphrase (empty for no passphrase):
    Enter same passphrase again:
    Your identification has been saved in ~/.ssh/id_ed25519_gal-terminus.
    Your public key has been saved in ~/.ssh/id_ed25519_gal-terminus.pub.
    The key fingerprint is:
    SHA256:6u8ZJ+5+5ne1gJ5M7Apb3K72RtPKn=Kgj74by3+zEp0 cluster_admin@hpc_clusterlab
    The key's randomart image is:
    +--[ED25519 256]--+
    |                 |
    |                 |
    |                 |
    |            E    |
    |        S....   .|
    |       ....o.. .+|
    |      .. o*..=.=o|
    |     . oO.--o.B+o|
```

(continues on next page)

(continued from previous page)

```
|     .=*=&@Osoo..|
+----[SHA256]-----+
```

and forward the public key (the content of `~/.ssh/id_ed25519_gal-terminus.pub`) by email to the Galactica database administrator. Don't worry, it IS safe to communicate the public key by email.

Configuring SSH target for Galactica

Configure a SSH target in your `~/.ssh/config` file with the SSH key file you just created to be able to upload data to the Galactica remote server:

```
Host Galactica_storage
Hostname galactica_ip_addr
User ssh_remote_user
Port ssh_port
IdentityFile ~/.ssh/id_ed25519_gal-terminus.pub
```

Here you can choose the target alias (here *Galactica_storage*) freely, you will need to provide it during the *Terminus configuration* step. The Galactica server IP address (*galactica_ip_addr*), ssh user (here *ssh_remote_user*) and SSH port (here *ssh_port*) will be provided to you by the Galactica database administrator, when he receives your SSH public key.

1.3 Terminus configuration

Once installed (see *Installation*), the `Terminus` package provides a helper command `terminus_config` designed to help you configure your *Terminus server instance*. This command will prompt you to answer some questions, some of them already answered in the previous section (see *Galactica access setup*), the others are detailed below.

Warning: Important The user who configures and runs the `Terminus` server **MUST** be the same user as the one who created the SSH key pair (see *SSH configuration*). For security reasons, it should be an unprivileged user.

1.3.1 First time configuration

The `terminus_config` command will create `.terminus` subdirectory (if not already present) into your `${HOME}` directory. This directory will contain the required configuration information to run your `Terminus` server. This command will guide you to define a few parameters (or hit `Enter` to use the default value) :

- **terminus_host_name** : name of the host of the `Terminus` server instance. By default the *hostname* of the machine, but can be set to a user-defined machine alias (name you gave to the Galactica web app. administrator in the section *Get RabbitMQ credentials*).
- **terminus_data_directory** : root directory path on the local filesystem where raw simulation data must be stored (see also *How to add raw simulation data on your Terminus server ?*).
- **terminus_job_directory** : directory path on the local filesystem where *post-processing job data directories* will be stored. For each job request received by your `Terminus` server, a temporary directory will be created at this location :
  - Upon failure, the *job data directory* will be moved to the `__failed` subdirectory,

  - Upon completion, the *job data directory* will be compressed and uploaded to the Galactica database server before cleanup.

- **terminus_service_directory** : directory path on the local filesystem where you must save all your custom post-processing service scripts (see also *How to create new post-processing services ?*).
- **terminus_number_of_nodes** : number of *workers* (processes) will be run on your `Terminus` server. It is usually recommended to use **1** *worker* per machine for this use case but you can increase it.
- **terminus_concurrency** : number of concurrent *threads* per *worker* that are running on your `Terminus` server instance. It depends on how much computational resources your are willing/authorised to allocate to simulation data post-processing through the Galactica web interface (you may discuss this particular topic with your head of lab., the machine sysadmin and/or the person who funded its purchase). It is recommended to set this parameter at a value that is not higher than the total number of CPUs available on that machine. Keep in mind that the maximum total number of concurrent data post-processing jobs that could be run by the `Terminus server` is :

  $N_{max} = terminus\_number\_of\_nodes \times terminus\_concurrency$

- **Galactica server SSH target** : SSH target name defined in your local ~/.ssh/config file that will be used to upload post-processed data back to the web application filesystem via a secure rsync link (see *Configuring SSH target for Galactica*).
- **SLURM_submission** : If SLURM is installed ont the machine then you can answer `Y` to this question and job submission will be done using the `sbatch` command. Otherwise answer `n`.
- The *RabbitMQ* credentials obtained previously (see *Get RabbitMQ credentials*) :
  - **RabbitMQ_user_name**, **RabbitMQ_host**, **RabbitMQ_port**, **RabbitMQ_virtual_host**. In addition, you need to edit the `~/.terminus/_secret/rabbitmq_server_pwd` to store the RabbitMQ user password in a secure way (file permissions are automatically set, you do not need to change them).

Note: File path auto-completion, tilde expansion and environment variable are available while typing values during the execution of the `terminus_config` command to help you out.

Finally addition, `terminus_config` inform the user of the configured user and group. In the case of a fresh install, the result of `terminus_config` command looks like:

```
$ terminus_config
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
# ~~~~~~~~~~~~~~~~~~~~~~ Terminus server configuration helper ~~~~~~~~~~~~~~~~~~~~~~ #
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #

        > Creating '/home/user1/.terminus' directory into the '/home/user1' directory.
        > Creating '_secret' directory into the '/home/user1/.terminus' directory.

Configuration of Terminus
Enter a terminus host name (current 'workstation'):
      > Default value used
Enter a terminus data directory (current '/home/user1'): /home/user1/simulation_data
Enter a terminus job directory (current '/home/user1'): /home/user1/terminus_jobs
Enter a terminus service directory (current '/home/user1'): /home/user1/terminus_services
Enter a terminus number of nodes (current 2): 1
Enter a terminus number of concurrency/node (current 2): 4
Enter a Galactica server SSH target name (configured in your local ~/.ssh/config file)␣
↪(current 'Galactica_storage'):
      > Default value used
```

(continues on next page)

(continued from previous page)

```
Use SLURM for job submission ? (Y/n) (current N): n

> Configured User: user1
> Configured Group: user1

------------------------------------ RabbitMQ configuration ----------------------------
↪---------------
Enter a rabbitmq username (current '__empty__')    : workstation_terminus
Enter a rabbitmq host (current '__empty__')        : 192.168.1.1
Enter a rabbitmq port (current '__empty__')        : 12541
Enter a rabbitmq virtual host (current '__empty__'): a_virtual_host

> Now set the provided RabbitMQ password in the file : /home/user1/.terminus/_secret/
↪rabbitmq_server_pwd
---------------------------------- RabbitMQ configuration completed ---------------------
↪---------------

Terminus configuration file written successfully.
```

The `.terminus` directory will contain the configuration files for Terminus :

- `terminus.env` : file containing environment variable required to run the `Terminus` server,
- `terminus.service` : systemd service config. file used for the deamonization of `Terminus` (see *How to launch Terminus workers as systemd deamon ?*),
- `_secret/rabbitmq_server_pwd` : file where you must store the RabbitMQ server password.

1.3.2 Update an old configuration

In the case of a subsequent call to the `terminus_config` command, parameter values you had set previously will be proposed to you as *default values* but your previous configuration will not be overridden before it is backuped :

```
$ terminus_config
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #
# ~~~~~~~~~~~~~~~~~~~~~ Terminus server configuration helper ~~~~~~~~~~~~~~~~~~~~~~~ #
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ #

Found a '.terminus' directory into the home directory

> terminus.env configuration file found !
  > backup to : '/local/home/user1/.terminus/terminus.env.27-10-2020-14-11-56'

> terminus.service configuration file found !
  > backup to : '/local/home/user1/.terminus/terminus.service.27-10-2020-14-11-56'

Terminus configuration loaded, ready to update.

Configuration of Terminus
Enter a terminus host name (current 'workstation'):
      > Default value used
Enter a terminus data directory (current '/home/user1/simulation_data'):
      > Default value used
```

(continues on next page)

(continued from previous page)

```
Enter a terminus job directory (current '/home/user1/terminus_jobs'): /home/user1/work_
↪terminus/jobs
Enter a terminus service directory (current '/home/user1/terminus_services'): /home/
↪user1/work_terminus/services
Enter a terminus number of nodes (current 1):
      > Default value used
Enter a terminus number of concurrency/node (current 4):
      > Default value used
Enter a Galactica server SSH target name (configured in your local ~/.ssh/config file)␣
↪(current 'Galactica_storage'):
      > Default value used
Use SLURM for job submission ? (Y/n) (current N): n

> Configured User: user1
> Configured Group: user1

...
```

1.4 Running Terminus server

1.4.1 How to launch Terminus workers as systemd deamon ?

The recommended setup is to run the *Terminus* server as a Systemd service :

```
> sudo systemctl enable ${HOME}/.terminus/terminus.service
    Created symlink from /etc/systemd/system/multi-user.target.wants/terminus.service to␣
↪/home/user1/.terminus/terminus.service.
    Created symlink from /etc/systemd/system/terminus.service to /home/user1/.terminus/
↪terminus.service.

> sudo systemctl start terminus.service
> sudo systemctl status terminus.service
     terminus.service
    Loaded: loaded (/home/user1/.terminus/terminus.service.; enabled; vendor preset:␣
↪enabled)
    Active: active (running) since Tue 2018-12-18 12:48:07 UTC; 6s ago
    Process: 3346 ExecStop=/bin/sh -c ${CELERY_BIN} multi stopwait ${CELERYD_NODES}    --
↪pidfile=${CELERYD_PID_FILE} (code=exited, status=0/SUCCESS)
    Process: 3781 ExecStart=/bin/sh -c ${CELERY_BIN} multi start ${CELERYD_NODES}    -A $
↪{CELERY_APP} --pidfile=${CELERYD_PID_FILE}    --logfile=${CELERYD_LOG_FILE} -loglevel=$
↪{CELERYD_LOG_LEVEL} ${CELERYD_OPTS} (
    Main PID: 3794 (python2)
        Tasks: 5
    Memory: 121.0M
        CPU: 972ms
    CGroup: /system.slice/terminus.service
            ├─3794 /home/user1/Terminus/python_env_1.0.0/bin/python2 -m celery worker --
↪time-limit=300 -A Terminus --concurrency=2 --loglevel=INFO --logfile=/home/user1/.
↪terminus/logs/w1.log --pidfile=/home/user1/.terminus/pids/w1.pid
            ├─3799 /home/user1/Terminus/python_env_1.0.0/bin/python2 -m celery worker --
```

(continues on next page)

(continued from previous page)

```
↪time-limit=300 -A webapp --concurrency=2 --loglevel=INFO --logfile=/home/user1/.
↪terminus/logs/w1.log --pidfile=/home/user1/.terminus/pids/w1.pid
           └─3802 /home/user1/Terminus/python_env_1.0.0/bin/python2 -m celery worker --
↪time-limit=300 -A webapp --concurrency=2 --loglevel=INFO --logfile=/home/user1/.
↪terminus/logs/w1.log --pidfile=/home/user1/.terminus/pids/w1.pid

   Jan 16 12:48:07 galactica-private systemd[1]: Stopped celery.service.
   Jan 16 12:48:07 galactica-private systemd[1]: Starting celery.service...
   Jan 16 12:48:07 galactica-private sh[3781]: celery multi v4.1.1 (latentcall)
   Jan 16 12:48:07 galactica-private sh[3781]: > Starting nodes...
   Jan 16 12:48:07 galactica-private sh[3781]:         > w1@galactica-private: OK
   Jan 16 12:48:07 galactica-private systemd[1]: Started celery.service.
```

As you can, see a log file for your worker can be found under the `logs` directory at `${HOME}/.terminus` so you have a trace of what is happening with the workers.

1.4.2 How to launch Terminus workers in *screen mode* ?

As an alternative, or for debugging purposes, you can also run the *Terminus* server in *screen* mode. In your case, you need to change *workstation* by your defined `hostname` :

```
> celery -A Terminus worker -c 4 -c:2 1 -Q workstation.terminus_jobs -Q:2 workstation.
↪monitor -l INFO -n worker1@%n monitor@%n

   -------------- worker1@workstation v4.1.1 (latentcall)
   ---- **** -----
   --- * ***  * -- Linux-3.10.0-514.21.1.el7.x86_64-x86_64-with-centos-7.3.1611-Core␣
↪2018-10-02 14:37:15
   -- * - **** ---
   - ** ---------- [config]
   - ** ---------- .> app:         Terminus:0x7fc90d7981d0
   - ** ---------- .> transport:   amqp://workstation_terminus:**@123.456.789.123:9876/
↪a_virtual_host
   - ** ---------- .> results:     disabled://
   - *** --- * --- .> concurrency: 4 (prefork)
   -- ******* ---- .> task events: OFF (enable -E to monitor tasks in this worker)
   --- ***** -----
   -------------- [queues]
                  .> workstation.terminus_jobs exchange=(direct) key=workstation.
↪terminus_job


   [tasks]
   . execute_terminus_job
   . update_terminus_job_status

   [2018-10-02 14:37:15,911: INFO/MainProcess] Connected to amqp://workstation_
↪terminus:**@**123.456.789.123:9876/a_virtual_host
   [2018-10-02 14:37:16,242: INFO/MainProcess] mingle: searching for neighbors
   [2018-10-02 14:37:17,715: INFO/MainProcess] mingle: sync with 2 nodes
   [2018-10-02 14:37:17,715: INFO/MainProcess] mingle: sync complete
   [2018-10-02 14:37:18,279: INFO/MainProcess] worker1@workstation ready.
```

1.5 Simulation data processing

1.5.1 How to add raw simulation data on your `Terminus` server ?

The first step is now to add raw simulation data on the Terminus server filesystem. The first question that needs to be answered is `where ?`

To avoid any confusion, the directory hierarchy within the `Terminus data directory` must match perfectly the end of the URL of the corresponding simulation page on Galactica, starting after `galactica-simulations.eu/db/`. Since that url is unique, it can be used to identify the target simulation data.

For example, the *ORION_FIL_MHD* simulation can be accessed at `http://www.galactica-simulations.eu/db/STAR_FORM/ORION/ORION_FIL_MHD/`. Hence the part of that URL that is going to be useful on Terminus is `STAR_FORM/ORION/ORION_FIL_MHD/`.The top directory is the *project catagory alias* (Star formation), the inner directory is the *projet alias* (Orion), and the child directory is the *simulation alias* (ORION_FIL_MHD).

To be accessed by Terminus data processing services, the raw data of that simulation must be stored within the `Terminus data directory`, following the exact same path : `${TERMINUS_DATA_DIR}/STAR_FORM/ORION/ORION_FIL_MHD/` . In this directory, you can transfer all the simulation snapshot directories (e.g. RAMSES outputs) See *Example* for an additional example.

New in version 0.7

To help the Terminus server administrator configure properly all these raw simulation data directories under the `Terminus data directory`, a `terminus_datasource_cfg` CLI tool is provided with the installation of Terminus. Upon simulation snapshot pages creation (and data association to post-processing service) on the Galactica simulation database web server (for projects hosted on your Terminus server), the Terminus server administrator will be notified of new data path definitions to configure on the Terminus server filesystem. It will take the form of a single JSON file that can be used as the only argument of the `terminus_datasource_cfg` command to automatically define the proper data paths and symbolic links :

```
> terminus_datasource_cfg new_datasource.json
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
↪~~~~~~~~~~~~~~ #
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~ Terminus datasource directory definitions ~~~~~~~~~~
↪~~~~~~~~~~~~~~ #
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
↪~~~~~~~~~~~~~~ #
Project category directory '/data/Terminus/STAR_FORM' already exists.
 -> Project directory '/data/Terminus/STAR_FORM/ORION' already exists.
     * Simulation directory '/data/Terminus/STAR_FORM/ORION/ORION_FIL_MHD' already␣
↪exists.
         - Created snapshot data symbolic link : '/data/Terminus/STAR_FORM/ORION/ORION_
↪FIL_MHD/output_00050' => '/raid/data/Proj_ORION/fil/MHD/output_00050'.
         - Created snapshot data symbolic link : '/data/Terminus/STAR_FORM/ORION/ORION_
↪FIL_MHD/output_00060' => '/raid/data/Proj_ORION/fil/MHD/output_00060'.
         - Created snapshot data symbolic link : '/data/Terminus/STAR_FORM/ORION/ORION_
↪FIL_MHD/output_00080' => '/raid/data/Proj_ORION/fil/MHD/output_00080'.
         - Snapshot data symbolic link unchanged : '/data/Terminus/STAR_FORM/ORION/ORION_
↪FIL_MHD/output_00120' => '/raid/data/Proj_ORION/fil/MHD/output_00120'.
         - Snapshot data symlink '/data/Terminus/STAR_FORM/ORION/ORION_FIL_MHD/output_
↪00150' has changed:
             + old target : '/raid/data/Proj_ORION/fil/Run4_phi3.45_beta0.5/output_00150
↪'.
             + new target : '/raid/data/Proj_ORION/fil/MHD/output_00150'.
```

(continues on next page)

(continued from previous page)

```
          Overwrite (y='yes', n='no', a='yes to all', x='no to all') [Y] ? y
          -> Deleted deprecated snapshot data symbolic link : '/data/Terminus/STAR_FORM/
↪ORION/ORION_FIL_MHD/output_00150' => '/raid/data/Proj_ORION/fil/Run4_phi3.45_beta0.5/
↪output_00150'.
        - Created snapshot data symbolic link : '/data/Terminus/STAR_FORM/ORION/ORION_
↪FIL_MHD/output_00150' => '/raid/data/Proj_ORION/fil/MHD/output_00150'.
Done...
# ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~
↪~~~~~~~~~~~~~~ #
```

Once this command is executed successfully, the Terminus can immediately access the configured simulation data, answer Galactica incoming job request and execute data processing jobs. No Terminus server restart is necessary.

1.5.2 How to create new post-processing services ?

First, go to your defined `terminus service directory` and create a new directory matching the name of the service (e.g. `my_service`). In that directory, copy and paste the following python template for your new service in a new python script, matching the name of the service (e.g. `my_service.py`):

```
${HOME}/terminus
└── services
    └── my_service
        ├── my_service.py
        └── service.json
```

```python
# -*- coding: utf-8 -*-
import os
import sys
#
# Do your custom imports here
import h5py  # If you want to export data as HDF5 file for example
#


def parse_data_ref(data_path, data_ref):
    """Fetch the required data

    :param data_path:
    :param data_ref: dataset reference as string
    :return:
    """
    #
    # Parse your data reference
    #
    # As an example, you could interpret the 'dataset reference' as a directory name, and␣
↪concatenates it
    # with the base data directory path
    dataset_path = os.path.join(data_path, data_ref)
    if not os.path.isdir(dataset_path):
        raise IOError("Simulation snapshot data directory not found.")

```

(continues on next page)

(continued from previous page)

```
    return dataset_path


def run(data_path, data_ref, argument_1="A", argument_2=256, argument_3=False,␣
↪test=False):
    """My custom data processing service description

    :param data_path: path to data directory (string)
    :param data_ref: dataset reference (string)
    :param argument_1:
    :param argument_2:
    :param argument_3:
    :param test: Activate service test mode
    """
    path = parse_data_ref(data_path, data_ref)

    if test:
        #
        # Run your service in test mode
        #
        return

    #
    # Python code for the service in normal mode goes here !
    #

    # The data processing service must write its output data in a local (already␣
↪created) "out/" directory.
    output_dir = "out"

    # Example => write data in a HDF5 file
    with h5py.File(os.path.join(output_dir, "results.h5"), 'w') as h5f:
        # Write result in a HDF5 dataset
        h5f.create_dataset("my_array", ...)


if __name__ == "__main__":
    run(sys.argv[1], sys.argv[2])

__all__ = ["run"]
```

Target data attributes

In the above script, the `run` python method will be directly called by the Terminus job upon execution. The **data_path** attribute is automatically transferred by Galactica during the job submission stage and matches the Galactica URL as explained in the previous section.

The **data_ref** attribute is the unique identifier of the simulation snapshot, as defined in Galactica. It allows to distinguish the various snapshots within a simulation and is transferred as a string (for example, the *data_ref* could be the snapshot number to post-process). Both the **data_path** and the **data_ref** attributes are provided to help you find the target data to run your service on (here in this example, they are parsed by the `parse_data_ref` function).

Test mode boolean attribute

The last attribute **test** is mandatory (set to False by default) and is used by Galactica to periodically test the Terminus data processing service. Indeed, Galactica will schedule periodic job requests in test mode (test=True) to monitor the availability of the Terminus server, the consistency of the data processing services, the existence of the target data and the correct execution of the processing service.

In *test* mode, the Terminus service will be run with the default values for the custom attributes (see next section), Galactica will not provide any value for these attributes. For performance reasons, you are advised to choose default values so that your service runs in the most degraded way (e.g. lowest resolution, small geometric region, ...) with these default attribute values.

Custom service attributes

The attributes between **data_ref** and **test** are specific to your data processing service and can be customized at will. The values for these attributes will be defined by the requesting user in the job submission form online. In this template we use only 3 attributes (argument_1, argument_2, argument_3) but you can use as many attributes as you need to perform your service.

You are encouraged to restrict the set of values considered valid for each parameter (e.g. value range for numeric attributes, limited set of possible resolution, restricted choice of physical quantity to process, etc.) and raise errors in case the received values are invalid. You can notify the Galactica administrator of these restrictions in order to design the job request submission form online accordingly.

The choice of the default values for your custom attributes is important: they must be set to run the service in a fast and memory-friendly way (see test mode section above).

Where the service must write data ?

During its execution, the data processing service will run in a dedicated job directory created in the *Terminus job directory* (see *Terminus configuration*) :

```
${HOME}/terminus
└── jobs
    ├── 148_density_map
    |    └── out
    └── 156_datacube3D
         └── out
```

within this job execution directory, an `out/` directory has been created for you tu put all the data files you want your service to provide. You only need to write data into this local `out/` directory within your service script and this directory will be tarballed upon job completion and finally uploaded to the Galactica server.

Service runtime configuration file

In addition to the service script file, Terminus requires a small JSON file named `service.json` that must contain the job **type** and **interpreter** required to run the service script. This allows, for example, the user to use python 2 service script while the Terminus server itself uses a python 3 environment, or even separate the python environments for each data processing service (each with their own dependencies).

Optionally, the job **n_nodes** (number of nodes), **n_cores** (total number of cores) and **timout_mn** (SLURM job timeout in minutes) integer parameters can be set (if you configured the Terminus server to submit processing jobs as SLURM jobs). These parameter values will define the SLURM submission parameters `#SBATCH -N n_nodes`, `#SBATCH -n`

`n_cores` and `SBATCH -t timeout_duration`. If left undefined, the default number of node is 1, the default total number of cores is 8, and the default SLURM job timeout 30 minutes.

The content of the `service.json` must look like this:

```
{
  "job":
    {
      "type":"python",
      "interpreter":"/path/to/interpretor/pythonX.Y",
      "n_nodes": 1,
      "n_cores": 16,
      "timout_mn": 30
    }

}
```

Publish your new post-processing service on the Galactica web app

Once this is done, **the final step** for the service to be available on Galactica, is to send some information to a Galactica admin :

- the name of your service and the name of the Terminus server on which you deployed it,
- a description of your data processing service, to document it on the Galactica web pages (full HTML content if you prefer, along with inserted images if required),
- the custom attribute names of the run method as well as their type and restrictions on their valid values. For example, **argument_1** must be an integer in the range [0,10], **argument_2** a character that can only take one of those values *[x,y,z]*, **argument_3** must a string chosen in the set *['gas density', 'gas temperature', 'gas metallicity']*, etc.

Once defined by the Galactica admin on the web server, you will be able to connect this new data processing service to your project so that authenticated users could submit job requests online to manipulate your raw data.

Note:

There is no need need to restart your `Terminus` server instance when :

- you create a new post-processing service script into your `terminus_service_directory`,
- you modify an existing service script into your `terminus_service_directory`,
- you activate a new `Terminus` post-processing service on Galactica.

1.6 FAQ

1.6.1 BUGS & SOLUTIONS

Error with submission script

Error `no display name and no $DISPLAY` with python scripts that use matplotlib:

While running a python script that use Matplotlib and try to save a figure on a server it is mandatory to use a non-GUI backend for Matplotlib even just to save the figure. Indeed, an error or a warning can been raised or simply the output file will not be created but the script will end successfuly.

```
Error: no display name and no $DISPLAY environment variable.
```

The solution is to add those lines at the top of the python script

```
import matplotlib
matplotlib.use("Agg")
```

A *userwarning* will be raised

```
/home/user1/works/venv/local/lib/python2.7/site-packages/matplotlib/figure.
↪py:457: UserWarning: matplotlib is currently using a non-GUI backend, so␣
↪cannot show the figure
"matplotlib is currently using a non-GUI backend, "
```

Error Terminus's daemonization

Error `-A option unrecognized` with Celery daemonization:

```
> journalctl -xe
  Error: no such option: -A
  oct. 28 15:24:12 Workstation sh[23120]: celery multi v5.0.0 (singularity)
  oct. 28 15:24:12 Workstation sh[23120]: > Starting nodes...
  oct. 28 15:24:12 Workstation sh[23120]:         > celery1@workstation: * Child␣
↪terminated with exit code 2
  oct. 28 15:24:12 Workstation sh[23120]: FAILED
  oct. 28 15:24:12 Workstation sh[23144]: > celery1@workstation: DOWN
  oct. 28 15:24:27 Workstation sh[23222]: Usage: __main__.py worker [OPTIONS]
  oct. 28 15:24:27 Workstation sh[23222]: Try '__main__.py worker --help' for help.
  oct. 28 15:24:27 Workstation sh[23222]: Error: no such option: -A

This is a reported error in Celery 5.0.0, that has been fixed `Celery_issue_6363`_.␣
↪Upgrade the celery to 5.0.1 fix that issue.
```

1.7 Example

Here we show an example of a directory tree structure with the `terminus data directory`, the `terminus service directory` and the `terminus job directory`. Let us assume we configured Terminus with those values (see *Terminus configuration*) :

```
> terminus data directory : /local/home/user1/terminus_example/data
> terminus job directory : /local/home/user1/terminus_example/jobs
> terminus service directory : /local/home/user1/terminus_example/services
```

```
${HOME}/terminus_example
├── data
|   └── GAL_FORMATION
```

(continues on next page)

(continued from previous page)

```
|       └── TEST_GALAXY
|           └── SIMU_1
|               └── output_00003
├── services
|   └── density_map
|       ├── density_map.py
|       └── service.json
└── jobs
    ├── 124_density_map
    |   └── out
    └── 126_density_map
        └── out
```

Let us assume that we published on Galactica a RAMSES simulation at the following url: https://authenticated.galactica-simulations.eu/db/GAL_FORMATION/TEST_GALAXY/SIMU_1 . The different simulation snapshots, if connected to any Terminus data processing service, must have their raw data located on the Terminus server filesystem within the same directory hierarchy as the Galactica web page url, assuming the configured Terminus data directory as the root of the hierarchy. In our example, the snapshot data directories must be stored in `/local/home/user1/terminus_example/data/GAL_FORMATION/TEST_GALAXY/SIMU_1` .

As for the data processing services, there is only one available service called `density_map` with the associated `density_map.py` python script. A JSON `service.json` file must be stored in the service directory.

The `density_map.py` has been written according to the script template given at the section *How to create new post-processing services ?*

```python
# -*- coding: utf-8 -*-
import matplotlib
matplotlib.use("Agg")

import sys
from pymses import RamsesOutput
from pymses.analysis.slicing import SliceMap
from pymses.analysis import Camera, ScalarOperator
from pymses.analysis.plot import PlainPNGImage, Plot2D

def run(data_path, data_ref, xcenter=0.5, ycenter=0.5, zcenter=0.5, depth=0.0, los_axis=
↪'z', up_vect='y'):
    """
    :param data_path: path to Ramses output directory (string)
    :param data_ref: output directory ref number (int)
    :param xcenter:
    :param ycenter:
    :param zcenter:
    :param depth:
    :param los_axis: line of sight
    :param up_vect: up vector
    :return:
    """

    if not os.path.isdir(data_path):
      raise IOError("Simulation data directory not found")
```

(continues on next page)

(continued from previous page)

```
    # Ramses outuput number
    ioutput = int(data_ref)
    ro = RamsesOutput(data_path, ioutput)

    # AMR data source
    amr = ro.amr_source(["rho"])

    # Defining a Camera object
    center = [xcenter, ycenter, zcenter]
    msize = 2**ro.info["levelmax"] # size defined to max level
    cam  = Camera(center, los_axis, region_size=[1., 1.], up_vector=up_vect, map_max_
↪size=msize)

    # Density field access operator
    rho_op = ScalarOperator(lambda dset: dset["rho"], ro.info["unit_density"])

    # Slice map computation
    slice_map = SliceMap(amr, cam, rho_op, z=depth) # create a density slice map at z=0.
↪5 depth position

    fname = "out/density_out-%d_%s" % (ioutput, los_axis)

    slice_map.save_HDF5(fname)

    color_map = "viridis"

    img_gen = PlainPNGImage(cmap=color_map, discrete=False, force_log_scale=False, alpha_
↪mask=True)
    slice_map.save_PNG(img_gen=img_gen, img_fname=fname+"-PNG.png")

    plot2D_gen = Plot2D( cmap=color_map, discrete=False, force_log_scale=False, axis_
↪unit=None, map_unit=None)
    slice_map.save_plot(plot_gen=plot2D_gen, plot_fname=fname+"-PLOT2D.png", fraction=0.
↪0, verbose=False)

__all__ = ["run"]
```

We sent by email the signature of the run function to Galactica admin

```
def run(data_path, data_ref, xcenter=0.5, ycenter=0.5, zcenter=0.5, depth=0.0, los_axis=
↪'z', up_vect='y')
```

as well as the range for the involved parameters: xcenter, ycenter, zcenter, depth varies between [0.0,1.0] and los_axis, up_vect can only takes one of those value {'x','y','z'}. The service is now available on the webpage of our simulation.

Once a job request is made, a directory will be created in the `terminus job directory` with the reference number of the request. As shown, in the directory tree above, we received two jobs requests for the service *density_map*, directory `124_density_map` and `126_density_map`. Once the job will be finished, the results of the job request which are in a directory named `out/` under the temporary job request directory will be compressed and sent to Galactica. And finally, those temporary job request directories will be removed.